UNIVERSIDADE DE LISBOA

Faculdade de Ciências

Departamento de Informática

DESENVOLVIMENTO E REENGENHARIA DE APLICAÇÕES WEB DE SUPORTE AO NEGÓCIO E INTEGRAÇÃO COM SISTEMAS DE BUSINESS INTELLIGENCE

Silvio Amir Alves Moreira

MESTRADO EM ENGENHARIA INFORMÁTICA

Engenharia de Software

2009

UNIVERSIDADE DE LISBOA

Faculdade de Ciências

Departamento de Informática

DESENVOLVIMENTO E REENGENHARIA DE APLICAÇÕES WEB DE SUPORTE AO NEGOCIO E INTEGRACAO COM FERRAMENTAS DE BUSINESS INTELLIGENCE

Silvio Amir Alves Moreira

PROJECTO

Trabalho orientado pelo Prof. Doutor Luís Alberto dos Santos Antunes

e co-orientado por Hélder Nuno Rodrigues Faria

MESTRADO EM ENGENHARIA INFORMÁTICA

Engenharia de Software

2009

Agradecimentos

Este trabalho representa o culminar de um longo processo de aprendizagem, formação e auto-conhecimento que contou com a intervenção de diversas pessoas das quais gostaria de mencionar algumas que, directa ou indirectamente, ofereceram inspiração e motivação, partilharam conhecimento e ajudaram a criar o, indispensável, suporte financeiro e emocional.

No âmbito do mestrado quero agradecer á Truewind por proporcionar todo espaço, liberdade criativa e acompanhamento que precisei, destacando o contributo do Hélder Faria pela energia e tempo pessoal que dedicou ao meu trabalho. Quero ainda agradecer ao Tiago Reis pela sua revisão desta tese.

Do percurso da licenciatura agradeço aos amigos e colegas de noitadas nos laboratórios pelas discussões, tertúlias e gargalhadas entre cafés. Sem esquecer os meus verdadeiros Amigos pelo apoio e compreensão nas minhas ausências (como podem ver "os projectos" existiam...). Não posso deixar de agradecer ao meu grande amigo e companheiro de guerra por ter tornado este caminho mais leve e, não poucas vezes, mais divertido. Obrigado Vilhena.

Finalmente agradeço á minha família, sem a qual nada disto seria possível, em especial á minha mãe que também queimou pestanas e fez as suas noitadas enquanto eu não chegava a casa. Obrigado minha mãe.

Para ti minha querida mãe.

Resumo

A crescente competitividade e exigência a nível de desempenho das organizações obriga a que estas, cada vez mais, tenham à sua disposição ferramentas de gestão que permitam monitorizar e acompanhar a saúde e desempenho do seu negócio no sentido de, por um lado, identificar, mitigar e evitar riscos atempadamente, e por outro, optimizar áreas problemáticas.

Apesar de existirem no mercado inúmeras ferramentas de suporte ao negócio com capacidades de *business intelligence*, estas tipicamente apresentam um carácter genérico, obrigando a um grande esforço de desenvolvimento, parametrização, instalação e manutenção de soluções, sendo construídas no âmbito de projectos de grande dimensão. Estes factores, aliados à utilização de modelos de desenvolvimento pouco ágeis, dão origem a soluções que apresentam dificuldade em acompanhar a evolução das necessidades e requisitos do negócio, comportando muitas vezes custos proibitivos para grande parte das empresas nacionais.

O trabalho que este relatório descreve consistiu: na reengenharia de uma aplicação previamente desenvolvida na organização de acolhimento que oferece a capacidade de cálculo de indicadores de desempenho, na maturação de alguns dos conceitos aplicados e na criação de um conjunto de peças de *software* que permitem, com baixos custos de desenvolvimento, instalação e manutenção oferecer soluções de *business intelligence,* à medida, em contextos e clientes diversificados.

O problema proposto foi abordado de uma perspectiva de engenharia de *software*, permitindo a identificação e implementação de um processo de desenvolvimento que permite optimizar custos de produção e melhorar a qualidade do produto final, o desenvolvimento de componentes reutilizáveis e a criação de uma arquitectura de *software* focada em atributos de qualidade, nomeadamente a extensibilidade, modificabilidade, interoperabilidade e desempenho do sistema.

Palavras-chave: *Business Intelligence*, Indicadores de Desempenho, Desenvolvimento Ágil, Engenharia de Software, Arquitectura de *Software*

Abstract

The increasing competitiveness and demand for continuous improvement of today's organizations, dictates the need for management tools that enable the monitoring and evaluation of their business health and performance. The analysis of these metrics will induce and allow the beforehand mitigation or elimination of these risks and also, the optimization of problematic processes, areas or sectors.

Despite the availability of innumerous business support tools with business intelligence capabilities currently available on the market, they tend to be complex and generic. As such, these tools normally require great effort in terms of development, customization, deployment and maintenance of solutions that are delivered within large and complex projects. These factors, allied to the use of non-agile development models and project management strategies, lead to solutions that have difficulty keeping in pace with the evolution of business needs and requirements and bear costs that are prohibitive to the majority of national companies.

The work described in this report consisted in the process of reengineering of an application that provides the extraction of key performance indicators, the maturation of some of the key concepts used in this solution and development of a set of software artifacts that, with low development, deployment and maintenance costs, allows for the delivery of custom business intelligence solutions in diverse customers and contexts.

The proposed problem was approached from a software engineering perspective that led to: the acknowledgment and implementation of a software development process that optimize production costs and increments the quality of the final product, the development of reusable components and the design of a software architecture focused in the quality attributes of the system namely its extensibility, modifiability, interoperability and performance.

Keywords: Business Intelligence, Key Performance Indicators, Agile Development, Software Engineering, Software Architecture

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Este relatório apresenta o trabalho concebido, desenhado, desenvolvido e avaliado, no âmbito do Projecto de Engenharia Informática. O seu objecto assenta na investigação e concepção de um sistema, baseado na reengenharia de aplicações *web* de suporte ao negócio, em tecnologia Microsoft, e na sua integração com ferramentas de *Business Intelligence*.

O primeiro capítulo apresenta a motivação, objectivos e enquadramento do projecto, tanto a nível organizacional como tecnológico.

No segundo capítulo é apresentado o planeamento e o trabalho de preparação e maturação dos conhecimentos necessários para levar a cabo o projecto proposto.

No capítulo três é documentada a pesquisa efectuada com o intuito de identificar um conjunto de metodologias, práticas e ferramentas adequadas ao paradigma da empresa na qual o projecto foi desenvolvido e aos objectivos do trabalho, sendo igualmente apresentado um resumo do processo adoptado.

No capítulo quarto são elencados os requisitos funcionais e de qualidade pretendidos para o produto desenvolvido.

No quinto capítulo são enumeradas as tácticas e princípios que orientaram o processo de desenho e é apresentada uma primeira aproximação à arquitectura de software que concretiza o sistema desenvolvido.

No sexto capítulo é feito o enquadramento tecnológico e são identificadas as ferramentas que suportam o processo adoptado e as tecnologias que concretizam o sistema.

O capítulo sete apresenta a reanálise e maturação da arquitectura de software proposta, motivada pela aplicação de padrões de desenho e por um melhor entendimento dos requisitos e das limitações impostas pela escolha de

tecnologias. Este capítulo é ainda caracterizado pela exposição técnica do fluxo de dados do sistema, apresentando, para cada fase, os conceitos e estratégias utilizados e demonstradas as medidas tomadas no âmbito da garantia de qualidade de software.

No capítulo oito são apresentados os resultados obtidos, quer a nível do processo, quer da solução desenvolvida, sendo explanado o modelo de desenvolvimento de soluções de *business intelligence* à medida, através do sistema concretizado. É ainda contextualizado o modelo de negócio que a ferramenta propicia e como este se enquadra nos serviços prestados pela empresa.

O nono capítulo faz uma avaliação da ferramenta desenvolvida, comparando qualitativa e quantitativamente o paradigma de soluções de *business intelligence* ágeis com o paradigma tradicional e comparando o sistema desenvolvido com a aplicação que serviu de base ao processo de reengenharia;

No último capítulo são apresentadas e discutidas as conclusões do trabalho realizado.

O trabalho descrito neste documento foi levado a cabo no âmbito de um estágio na Truewind, Tecnologias de Informação, SA (Truewind), na área de Análise e Desenvolvimento de Sistemas de Informação, dentro da Unidade *Microsoft*. A Truewind, é uma empresa com forte competência nas áreas de desenvolvimento de software, especializada na identificação e fornecimento de soluções tecnológicas centradas nas necessidades dos seus clientes, utilizando metodologias ágeis, e na prestação de serviços de suporte a sistemas de informação baseada em contrato – *managed services*. Estes serviços desenvolvem-se sobre três vectores principais: consultoria em tecnologias de informação;análise e desenvolvimento de sistemas de informação; serviços de suporte e manutenção de sistemas e bases de dados, abarcando as seguintes áreas de competência tecnológica unidade *Java*; unidade *OutSystems*; unidade *Microsoft* e unidade *Oracle*.

1.1 Motivação

A consolidação das infra-estruturas de recolha da informação nas organizações em suites de aplicações horizontais – *Enterprise Resource Planning (ERP)* – é hoje uma realidade para organizações de média e grande dimensão. Contudo, e apesar destas suites de aplicações incluírem muitas vezes ferramentas de *Business Intelligence*, estas, apresentam um carácter genérico e nem sempre são extensíveis e capazes de responder à globalidade do parque

aplicacional das organizações e de integrar e relacionar informação proveniente de diferentes fontes de dados.

Limitar a abrangência da análise do desempenho operacional de uma organização apenas a partes do seu negócio não promove a decisão apoiada em factos objectivos, deixando graus de liberdade que podem ter consequências na tomada de decisões. É pois necessário apostar na normalização e na reengenharia de aplicações relevantes para o negócio de forma a que, tanto do núcleo aplicacional suportado no *ERP*, como das aplicações periféricas, seja possível extrair e integrar indicadores de desempenho. Com base nesta premissa, é possível obter a integração plena de sistemas de *Business Intelligence* com as várias aplicações de negócio para o cálculo dos *Key Performance Indicators (KPI)* de uma organização, auxiliando a tomada de decisões estratégicas seguras e informadas.

Tomando como ponto de partida uma solução desenvolvida especificamente para um cliente da Truewind, desenvolveu-se uma plataforma tecnológica que permite, de forma simples, rápida e com baixo custo de desenvolvimento e instalação, disponibilizar soluções de *Business Intelligence* criadas à medida, trazendo valor através do profundo conhecimento que a Truewind adquiriu sobre o negócio dos seus clientes. Esta plataforma enquadra-se no paradigma de negócio da empresa, numa lógica de prestação de *managed services*, na medida em que sendo facilmente adaptável e extensível, permite acompanhar a evolução do negócio do cliente, oferecendo sempre em cada momento a informação considerada mais relevante para a avaliação do desempenho da organização e para a tomada de decisões operacionais e estratégicas.

1.2 Objectivos

O objectivo de partida deste projecto consistiu na reengenharia de um produto pré-existente, o SicGest, tendo, no entanto, sido dado um foco igualmente importante à identificação e definição de um processo de desenvolvimento que permita incrementar a qualidade do produto a desenvolver mas que seja suficientemente genérico para poder ser reutilizado em futuras soluções.

Relativamente ao processo de desenvolvimento pretendeu-se, no contexto da utilização de metodologias ágeis, investigar e aplicar as melhores práticas de engenharia de software para montar um processo estruturado que permita optimizar quer custos de desenvolvimento, quer a qualidade do produto. Com

este intuito foram identificadas e analisadas um conjunto de ferramentas de suporte a estas práticas. Ao longo do desenvolvimento da solução garantiu-se a utilização de boas práticas de desenho e execução e foram aplicados os padrões arquitecturais e de desenho adequados aos objectivos funcionais e qualitativos.

No que diz respeito ao produto desenvolvido, o principal objectivo incidiu no redesenho de uma aplicação de extracção e visualização de indicadores de desempenho, escrita numa tecnologia obsoleta (*ASP* 3.0) e desenvolvida especificamente para um cliente. Reaproveitaram-se alguns dos conceitos base utilizados, tendo sido desenvolvida uma nova solução, baseada nas plataformas de desenvolvimento mais actuais disponibilizadas pela Microsoft e num esforço de investigação e desenvolvimento. Utilizando as ferramentas e abordagens identificadas foi atingido o objectivo de permitir de forma ágil e com baixos custos de desenvolvimento e instalação, disponibilizar soluções de *business intelligence* em contextos e clientes diversificados.

1.3 Integração na empresa

O projecto iniciou-se a 6 de Outubro de 2008 com uma fase de integração na empresa, que decorreu nas instalações da Truewind e num dos seus principais clientes. Esta fase teve como principais objectivos conhecer a estrutura e organização da empresa, as diferentes unidades de negócio e os seus métodos de trabalho. Ao mesmo tempo foi iniciada a formação específica em tecnologias *Microsoft*, nomeadamente no desenvolvimento de soluções de portais intranet e corporativos suportados em *Microsoft Sharepoint* com vista à familiarização com esta tecnologia. Este período de formação, possibilitou igualmente a análise da possibilidade de integração da tecnologia de indicadores de desempenho a desenvolver em portais desta natureza com o objectivo de disponibilizar no contexto do ambiente de trabalho dos utilizadores finais, informação sobre o desempenho da sua organização.

1.4 SicGest

Um dos principais objectivos do trabalho efectuado, consistiu no redesenho de uma aplicação desenvolvida no âmbito de um projecto anterior da Truewind, que permite a extracção e visualização de indicadores de desempenho calculados a partir do repositório de dados de negócio – o **SicGest (**Figura 1**)**. Esta aplicação tem uma infra-estrutura extremamente simples que consiste num conjunto de procedimentos de bases de dados (*stored procedures*) que periodicamente: lêem um conjunto de *queries* previamente parametrizadas que concretizam um conjunto de indicadores; efectuam a sua execução e armazenam os resultados em tabelas que servem como um repositório local de dados. Quando o *dashboard* é apresentado, são disponibilizados os valores pré-calculados dos indicadores, sendo os resultados disponibilizados em diversos formatos de acordo com a sua natureza, nomeadamente, através de tabelas, gráficos circulares, de barras ou de linhas. Para representações sob a forma de tabela, os valores dos indicadores podem ter associados *links* para aplicações específicas que permitem obter mais detalhes sobre o valor de um dado indicador.

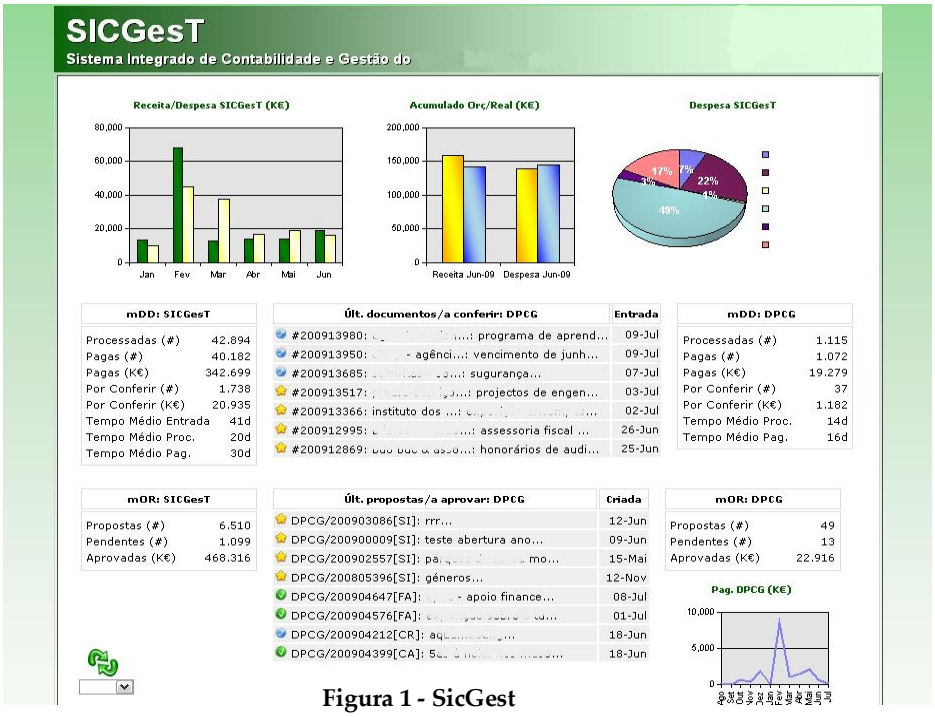


Figura 1 - SicGest

Apesar da sua simplicidade, esta aplicação demonstrou ser uma importante ferramenta de gestão, tendo granjeado um significativo sucesso junto do cliente, um organismo da administração pública. Contribuiu ainda, de forma decisiva, para o reconhecimento do projecto no qual se inseria, no âmbito de prémios de boas práticas no sector público.

Capítulo 2

Planeamento

2.1 Mapa de Gantt

Na Figura 2, é apresentado o calendário de execução do Projecto de Engenharia Informática decorrido na Truewind.

ID	Task Name	Duration
0	**Projecto Engenharia Informática**	**48 wks**
1	**Integração na Empresa**	**4 wks**
2	Integração na empresa e maturação proposta de estágio	2 wks
3	Análise ferramenta SICGEST	1 wk
4	Pesquisa ferramentas KPI / BI análogas	1 wk
5	**Enquadramento**	**12 wks**
6	Inclusão em projectos em curso	10 wks
7	Consolidação de práticas metodológicas e tecnologias utilizadas	2 wks
8	**TrueKPI - Análise**	**11 wks**
9	Definição de requisitos funcionais e atributos de qualidade	2 wks
10	Definição da arquitectura inicial	2 wks
11	Investigação e análise das metodologias, técnicas e processos	3 wks
12	Escolha ferramentas de suporte às metodologias a utilizar	3 wks
13	**TrueKPI - Desenvolvimento**	**13 wks**
14	Sprint 0 - Construção do ambiente de desenvolvimento	2 wks
15	Sprint I - Criação de Indicadores	2 wks
16	Sprint II - Obtenção de indicadores	2 wks
17	Redifinição da arquitectura	2 wks
18	Sprint III - Visualização de indicadores	2 wks
19	Sprint IV - Gestão e visualização de indicadores noutras plataformas	2 wks
20	Casos práticos	1 wk
21	**Resultados e Relatório**	**5 wks**
22	Recolha Resultados	1 wk
23	Relatório	4 wks

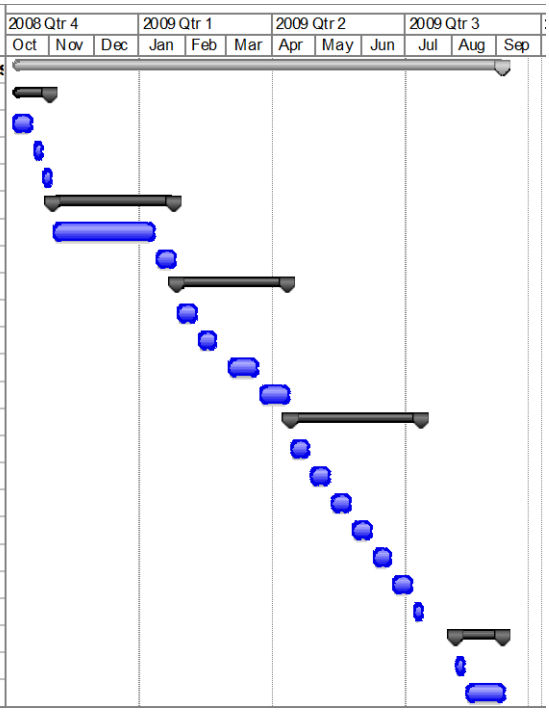


Figura 2 - Mapa de Gantt de execução do projecto

O trabalho desenvolvido no âmbito do estágio desenrolou-se nas seguintes etapas:

- **Integração na Empresa** – Nesta fase o aluno teve oportunidade de conhecer em maior detalhe a forma de organização da empresa de acolhimento, as suas áreas de actuação e alguns dos projectos em que se encontra envolvida; utilizar e analisar a ferramenta, previamente existente, que esteve na base do trabalho que foi desenvolvido; e de pesquisar e analisar outras ferramentas

comerciais de alguma forma enquadradas no propósito do SicGest e do projecto a desenvolver.

- **Enquadramento Tecnológico e Metodológico** – Como forma de amadurecer as suas competências tecnológicas, adquirir e conhecer as práticas metodológicas utilizadas pela empresa e de fomentar a sua integração na equipa ligada à área de interesse do projecto, o aluno foi integrado em vários projectos em curso num dos clientes da Truewind; na sequência deste período, teve oportunidade de consolidar e estruturar as práticas metodológicas, de forma a suportar as fases de análise e desenvolvimento do projecto, e ainda de validar a escolha da plataforma tecnológica.
- **TrueKPI – Análise** – Este período centrou-se em quatro actividades principais: definição de requisitos funcionais e de atributos de qualidade que deveriam ser assegurados pela ferramenta a desenvolver; desenho e definição da arquitectura inicial da solução; aprofundamento e escolha, numa perspectiva da engenharia de software, da metodologia, das tácticas e dos processos, que por um lado, fossem adequados ao desenrolar do projecto, e por outro, optimizassem a execução futura de projectos baseados na ferramenta a desenvolver; pesquisa e selecção de um conjunto de ferramentas de suporte ao processo de desenvolvimento e princípios definidos.
- **TrueKPI – Desenvolvimento** – Seguindo a metodologia *scrum*, a fase de construção da solução foi efectuada de forma iterativa e incremental. Esta fase foi organizada num conjunto de ciclos ou *sprints*, nos quais foram sendo adicionadas e validadas novas funcionalidades à ferramenta, cobrindo cada um dos requisitos e atributos definidos na fase de análise; no decorrer deste processo foi dedicado um dos ciclos intermédios à redefinição e validação da arquitectura da solução. Como corolário desta fase de desenvolvimento, foi utilizada a ferramenta construída para a construção de um *dashboard* de indicadores análogo ao suportado pelo SicGest, tirando partido das novas funcionalidades e características suportadas.
- **Resultados e Relatório** – Como fase final do estágio, procedeu-se à recolha e análise de resultados e à elaboração do presente relatório.

Capítulo 3

Processo e Metodologias

Capitalizando o grande grau de liberdade oferecido ao aluno na gestão e execução do seu trabalho e sabendo que o Projecto Engenharia Informática, pode e deve ser usado também como espaço de experimentação de ideias e conceitos, o aluno aproveitou a oportunidade para investigar e experimentar algumas metodologias, técnicas e ferramentas de Engenharia de Software em fase de adopção na empresa.

Na Truewind não está instituído um único processo formal de desenvolvimento, permitindo e levando a que em cada projecto sejam utilizados os processos, metodologias e ferramentas mais adequados à dimensão, custo, risco, recursos disponíveis e necessidades de cada caso concreto. Apesar da proposta de estágio estar centrada no desenvolvimento de um produto, o aluno estando a especializar-se na área de Engenharia de *Software* deu grande enfoque não só ao produto, mas também ao seu processo de desenvolvimento. Para esse efeito, foi efectuado um trabalho de pesquisa sobre as melhores práticas e ferramentas para conduzir este processo, permitindo que o projecto realizado neste estágio servisse como prova de conceito para avaliação da utilização destas técnicas, práticas e ferramentas, contribuindo desta forma para a consolidação desta base de conhecimento na empresa. Nas secções seguintes são descritas as metodologias, técnicas e ferramentas analisadas e utilizadas e o processo de desenvolvimento do trabalho.

3.1 Metodologias

3.1.1 *Scrum*

A engenharia de *software* é uma actividade de gestão de conhecimento que tem que lidar com múltiplas incertezas, sejam elas tecnológicas, de negócio, sociais ou outras. Todas estas incertezas aumentam os riscos dos projectos, levando a situações em que, para se mitigarem estes riscos, são adoptadas metodologias que atrasam significativamente o *breakeven* do projecto (seja por via da elaboração inicial de documentos de especificação muito detalhados, seja por via de atrasos no processo de desenvolvimento provocados por erros ou omissões nos documentos de especificação, seja por erros de estimação detectados em fase tardias do projecto).

A generalidade das metodologias em uso resiste à mudança, procurando adiar todas as alterações ao projecto para depois da execução do mesmo. Esta estratégia é diametralmente oposta às necessidades do negócio: ao longo do desenvolvimento do projecto, o negócio quer continuar a alterar métodos e procedimentos para aumentar a competitividade da organização.

O *Scrum* é uma metodologia incremental e iterativa de gestão de projectos, que encapsula as práticas de desenvolvimento de software usadas, focada em criar valor para o cliente de forma imediata e contínua; é principalmente indicada para projectos com um grau de incerteza elevado, com requisitos pouco definidos ou com grande susceptibilidade de mudança.

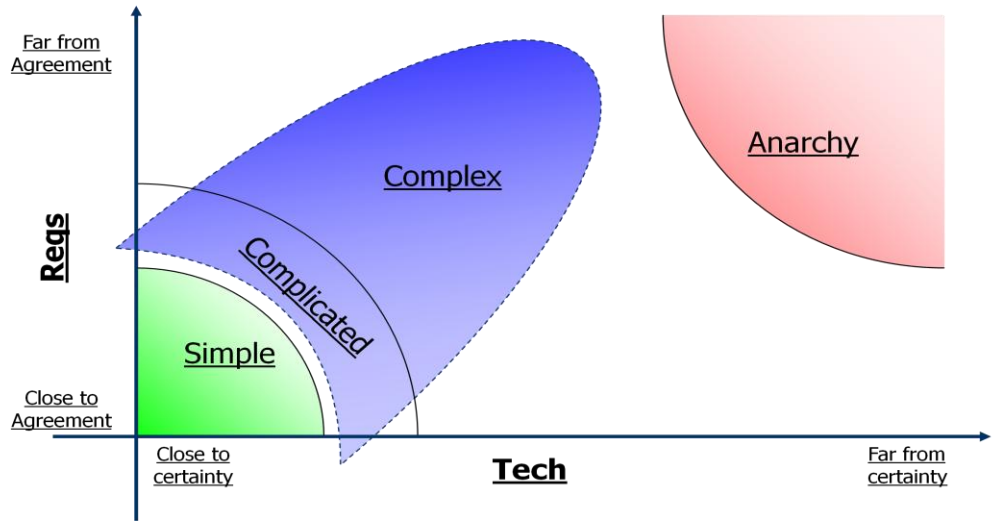


Figura 3 - Complexidade de projectos

Esta metodologia propõe que o projecto seja estruturado em ciclos de trabalho relativamente curtos – tipicamente 2 a 4 semanas –, denominados *sprints*, em que é feita a implementação de um conjunto de funcionalidades originando uma versão funcional do produto que pode ser usada para avaliação por parte do cliente ou até num ambiente de produção. Desta forma é possível obter o contributo imediato por parte do cliente, alinhar expectativas e ajustar o planeamento fazendo uso de métricas recolhidas nos *sprints* anteriores. A metodologia propõe a criação e manutenção de uma lista de requisitos funcionais, sob a forma de *user stories*, requisitos de qualidade, itens de trabalho e *bugs*, esta lista deverá ser prioritizada com o *product owner* (tipicamente o cliente) de forma a que as funcionalidades mais importantes e/ou que geram mais valor sejam desenvolvidas de antemão. Esta lista denominada *Product Backlog* é pública, dinâmica e pode ser alimentada em qualquer altura e por qualquer dos intervenientes. Os *sprints* são construídos a partir desta lista seleccionando um conjunto de entradas para formar uma sub-lista chamada *Sprint Backlog*, esta lista é privada à equipa responsável pelo *sprint* e não poderá ser alterada até ao final desse ciclo de desenvolvimento. No *scrum*, a análise de requisitos funcionais é feita a partir de *user stories*, uma *user story* é uma frase simples que deve especificar uma funcionalidade do sistema mas explicada do ponto de vista do utilizador. Uma *user story* deve focar-se no ***porquê***, ***quem*** e ***o quê*** de uma funcionalidade e não no ***como***, não deve ter um carácter técnico e deve estar escrita na linguagem do domínio.

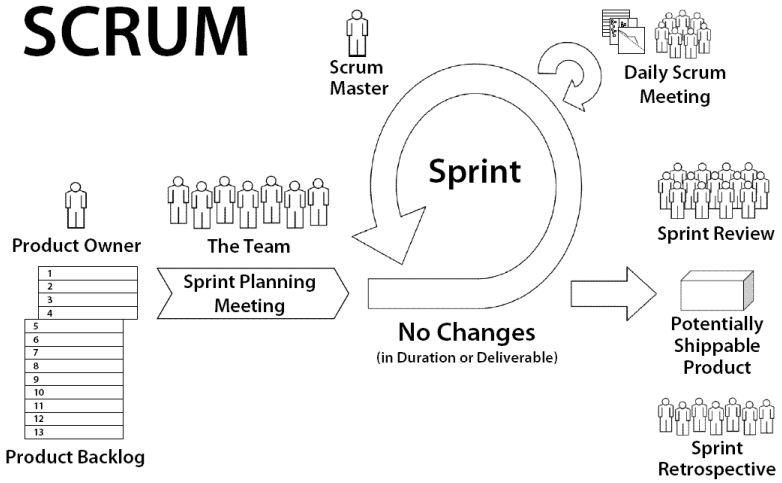


Figura 4 - Ciclo de gestão *Scrum*

O *Scrum* tem vantagens sobre as metodologias tradicionais de gestão de projectos porque, por um lado, a sua filosofia de iterações e *feedback* constante faz com que seja mais resiliente a mudanças nos requisitos e no âmbito do projecto e, por outro lado, tem um conjunto de características que facilitam a

gestão e mitigação dos principais riscos que levam a que os projectos falhem, nomeadamente:

- **Risco de não cumprir as expectativas do cliente**: mitigado através de entregas e *feedback* frequentes;
- **Risco de planeamento e estimativas erradas**: mitigado através de estimação e prioritização constante;
- **Risco de não conseguir resolver os problemas de forma célere**: mitigado através de reuniões diárias de acompanhamento;
- **Risco de não conseguir entregar o produto prometido**: mitigado através de entregas frequentes;
- **Risco de aceitar compromissos com objectivos demasiado ambiciosos**: mitigado através da definição de objectivos específicos e âmbito delimitado em cada *sprint*;

3.1.2 *Attribute Driven Design*

Attribute Driven Design é uma abordagem utilizada para obter uma arquitectura de software que satisfaça um conjunto de atributos funcionais e de qualidade. Esta abordagem aceita como entrada os requisitos de qualidade que o sistema deve exibir, sob a forma de cenários que especificam esses atributos. O processo consiste em decompor o sistema de forma recursiva e, em cada passo, aplicar tácticas e padrões arquitecturais que permitam atingir os objectivos propostos. Este processo pode iniciar-se logo no início do ciclo de desenvolvimento, a partir do momento em que sejam conhecidos os principais requisitos funcionais e atributos de qualidade a atingir e vai sendo refinado ao longo do processo de desenvolvimento de forma a contemplar um melhor entendimento dos requisitos e do domínio, bem como, as mudanças e restrições impostas por estes requisitos ou pela tecnologia adoptada.

3.1.3 Integração Contínua

A integração contínua é uma das práticas propostas pela metodologia *eXtreme Programing* [1] e visa facilitar o processo de integração de código, propondo que a inclusão de novas funcionalidades seja feita de forma contínua. Tradicionalmente, um projecto é desenvolvido por um conjunto de programadores, sendo que cada um fica responsável por desenvolver uma parte do código; no final será necessário juntar todas as peças para formar o produto.

Esta abordagem pode ser pouco eficiente uma vez que podem existir falhas no *software* que são introduzidas no processo de integração, ou seja o facto de todos os módulos estarem correctos não garante a correcção de um sistema composto por estes, uma vez que podem surgir erros nas suas interacções. Este facto é agravado pela dificuldade de detecção e correcção de uma falha aumentar com a complexidade da base de código e com a quantidade de desenvolvimentos que foram feitos sobre uma base de código errónea. Outro problema desta abordagem é a dificuldade em estimar o tempo necessário para a fase de integração (que pode demorar meses no caso de projectos grandes) e saber em dado momento que percentagem do esforço está concluído. Consequentemente, é mais difícil estimar a data de conclusão do projecto.

A integração contínua propõe que exista um repositório central, partilhado por toda a equipa com a versão actual do código e que cada membro da equipa de desenvolvimento faça integrações frequentes do novo código desenvolvido (pelo menos uma integração por dia). O processo deverá ser automatizado através de *scripts* que consigam recriar o produto por completo, validem a integração e verifiquem a integridade da versão através de testes automáticos de forma a detectar erros o mais rapidamente possível, diminuindo assim o seu custo de detecção e correcção. No limite deve ser possível recuar para a versão anterior do código repondo um estado correcto do produto a desenvolver.

Desta forma o repositório tem sempre uma versão estável e com todas as funcionalidades que vão sendo integradas, diminuindo os ciclos de desenvolvimento e permitindo logo numa fase inicial do projecto gerar valor para o cliente e/ou obter *feedback* deste e assim estar sempre alinhado com as suas necessidades e expectativas.

O ciclo de desenvolvimento, esquematizado na Figura 5, consiste nos seguintes passos:

1. O programador obtém a versão mais recente do código do sistema de controlo de versões (através de um ***Update***[1] ou ***Check-Out***[2]);
2. O programador desenvolve as novas funcionalidades e a bateria de testes que faz a verificação e validação das funcionalidades;
3. O programador valida que todas as funcionalidades estão implementadas e todos os testes são executados com sucesso;

[1] Comando que actualiza o código fonte com a versão mais recente a partir do repositório
[2] Comando que obtém a versão mais recente do código fonte

4. O programador faz ***Commit***[3] no sistema de controlo de versões (do código funcional e dos novos testes);
5. O servidor detecta alterações na base de código do sistema de controlo de versões e actualiza o código na área de trabalho (através de ***Update***, caso já tenha uma versão do projecto ou ***Check-Out***, caso contrário);
6. O servidor gera uma nova versão do produto e executa toda a bateria de testes;
7. O servidor gera relatórios de actividade.

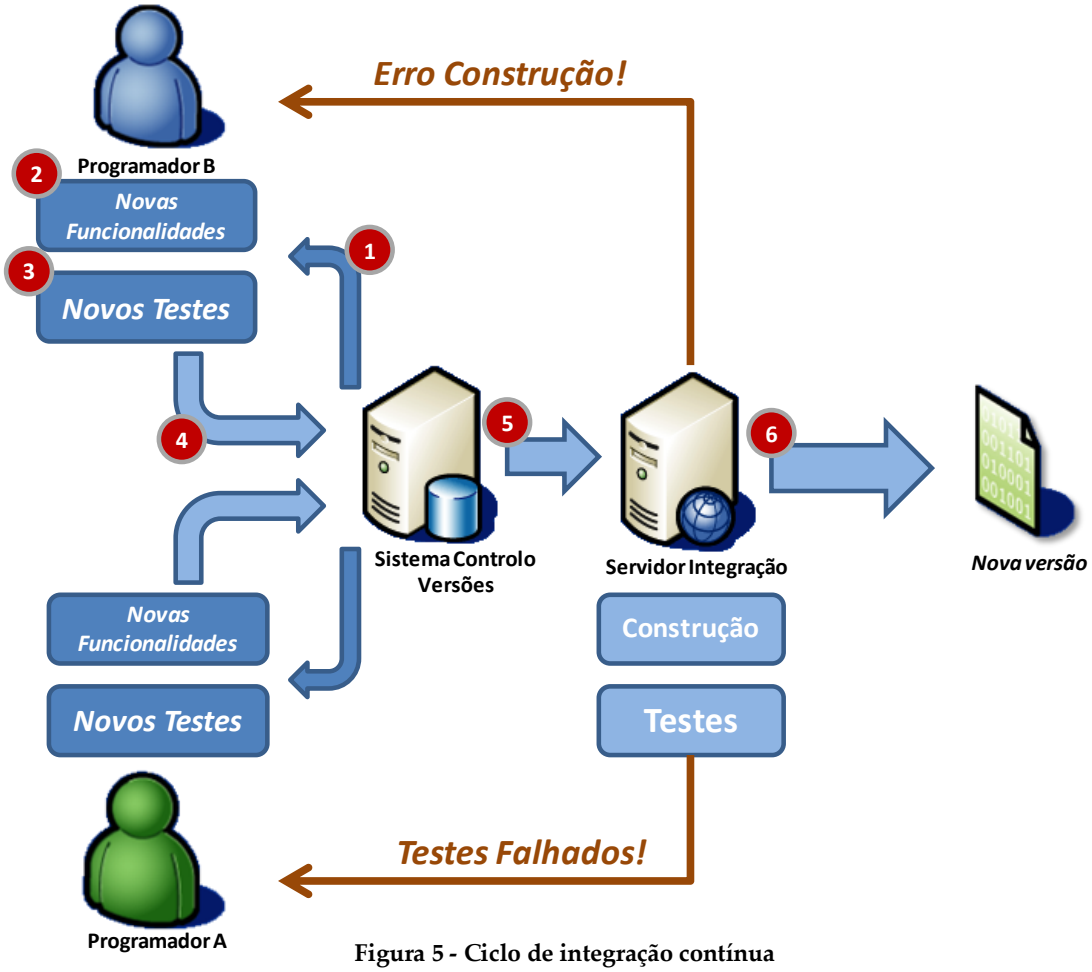


Figura 5 - Ciclo de integração contínua

Caso seja possível gerar uma nova versão e todos os testes sejam executados com sucesso, o processo termina, caso contrário o programador tem de corrigir os erros e voltar ao passo 3.

[3] Comando que actualiza o repositório de versões com o código desenvolvido.

3.2 Processo

O processo de desenvolvimento do trabalho efectuado pode ser resumido em 3 fases: uma fase inicial de análise e contextualização, uma fase de investigação, definição de processos e metodologias e finalmente a fase de execução do trabalho.

O desenvolvimento do trabalho iniciou-se com uma análise detalhada da proposta e da aplicação existente com o intuito de perceber os requisitos de alto nível, o trabalho a realizar e as expectativas face ao produto a desenvolver. Esta fase terminou com um esboço da infra-estrutura e de uma versão inicial do modelo de dados adequado para suportar as funcionalidades pretendidas.

A segunda fase consistiu na investigação e análise de metodologias e técnicas para agilizar e optimizar o desenvolvimento do trabalho proposto, com o potencial de serem integradas nas práticas em adopção na empresa. Esta fase culminou com a adopção de uma abordagem para o desenho da arquitectura de *software* e um método para a documentação da mesma e dos atributos de qualidade. Esta metodologia, denominada *Attribute Driven Design*, leva em conta não só os requisitos funcionais, mas também os de qualidade na fase inicial do desenho da arquitectura, e utiliza um conjunto de padrões e tácticas para atingir estes objectivos. Desta forma, garante-se que estes atributos são considerados e incorporados na arquitectura, uma vez que a experiência diz-nos que considerar os aspectos de qualidade numa fase tardia do desenvolvimento pode trazer consequências nefastas, que podem passar por obrigar a um redesenho da arquitectura (com todos os custos inerentes), pela eliminação de algumas funcionalidades do sistema ou, no limite, não ser possível garantir alguns dos atributos ou requisitos fundamentais.

O processo de desenvolvimento de software adoptado foi a integração contínua com testes unitários automatizados. Esta abordagem permite ciclos de desenvolvimento mais curtos com produção imediata de valor para o cliente, o que faz com que seja um bom complemento à metodologia de gestão de projectos adoptada (*Scrum*). Permitiu aumentar a qualidade do produto através da existência de uma infra-estrutura de testes que valida cada uma das funcionalidades e a integração destas no produto final. Esta infra-estrutura de testes induz a redução do esforço de desenvolvimento, no sentido em que possibilita encontrar erros numa fase inicial do processo e identificar que integração foi responsável pela introdução dos mesmos. Isto permite, por um lado, repor se necessário o estado correcto do produto, retirando as últimas funcionalidades integradas e por outro definir um âmbito circunscrito para a

depuração e correcção dos erros, reduzindo o tempo necessário para efectuar estas correcções.

Sabendo que uma parte considerável do tempo de desenvolvimento num projecto é gasto em depuração e correcção de erros [2], percebemos que a aplicação desta metodologia pode optimizar a sua fase de construção. Outra vantagem que a integração contínua com testes automatizados oferece, é a de facilitar o processo de refactorização aumentando o nível de confiança dos programadores, no sentido em que os testes validam as alterações feitas de forma a não quebrar o bom funcionamento do produto.

Após a definição das metodologias e processos a utilizar, foi feita uma pesquisa com o intuito de identificar e testar as ferramentas de suporte ao processo definido, tendo-se construído um ambiente de desenvolvimento que as englobasse. A fase final, ou de execução, do projecto assentou nas metodologias supracitadas e iniciou-se com a definição de um *product backlog* resultante de uma análise detalhada dos requisitos, tendo-se desenrolado de forma iterativa numa lógica de prototipagem sucessiva, em que cada nova versão acrescenta novas funcionalidades à anterior. Dentro de cada uma das iterações (*sprints*), as funcionalidades foram acrescentadas de forma sucessiva e contínua, validando sempre a integridade do produto no final de cada integração.

Capítulo 4

Requisitos

Tradicionalmente, as aplicações de *business intelligence* são utilizadas por grandes empresas para optimizar o seu desempenho com suporte em decisões estratégicas e operacionais informadas. Com a evolução e maturação dos modelos de negócio e o incremento da competitividade em todos os sectores de actividade, é hoje em dia indispensável às empresas e organizações de qualquer sector de actividade ter a capacidade de avaliar a sua produtividade, desempenho e encontrar pontos de optimização. O objectivo da ferramenta desenvolvida, intitulada ***TrueKPI,*** é oferecer uma solução de baixo custo que ofereça as funcionalidades de cálculo de indicadores de negócio e que permita a sua disponibilização de forma facilitada e integrada no ambiente de trabalho dos utilizadores finais, consubstanciado num portal *intranet*.

Este capítulo apresenta os requisitos definidos para esta aplicação, estando organizado em duas secções: requisitos funcionais, isto é, as funcionalidades que deverão ser suportadas, traduzidas sob a forma de *user stories* e requisitos de qualidade, as propriedades não funcionais que a aplicação deverá exibir.

Os requisitos foram elencados através da análise da proposta de estágio, da análise da versão previamente existente da aplicação e com o contributo de alguns elementos mais experientes da empresa, quer no início do estágio, quer posteriormente através da sua revisitação e revisão ao longo de todo o projecto, nomeadamente no inicio de cada novo *sprint*.

4.1 *User Stories*

Os requisitos funcionais, listados na Tabela 1, foram documentados utilizando a notação proposta pela metodologia *Scrum* – as *user stories*. As *user stories* devem capturar os requisitos através de pequenas histórias que descrevem uma funcionalidade, e são constituídas por 3 secções:

- **Cabeçalho**: Deve conter um identificador;
- **Conversa**: Deve registar a essência do requisito, descrito sob a perspectiva do utilizador e utilizando linguagem não técnica e específica do domínio. Estas histórias podem ser escritas no formato:
    - "Eu como [*Papel*] posso [*Função*] de forma a [*Lógica*]";
    - Uma frase simples que ilustre um cenário de utilização;
    - Uma palavra, se for suficientemente descritiva;
- **Validação**: Casos de teste para validar o requisito. A escrita de testes na fase de levantamento de requisitos permite obter uma melhor compreensão dos mesmos por parte do cliente e da equipa de desenvolvimento e antecipar cenários de utilização que, doutra forma, poderiam ser detectados apenas numa fase tardia do processo.

Todos os detalhes técnicos e da implementação são discutidos *a posteriori* pela equipa na fase de desenvolvimento.

Cabeçalho	Conversa	Validação
US1	Como utilizador consigo construir um novo indicador, especificando as suas propriedades	Criar um novo indicador de desempenho e validar que este é correctamente armazenado
US2	Como utilizador posso consultar indicadores através uma página *web* ou portal *sharepoint*	Visualizar os indicadores criados
US3	Como utilizador, consigo editar ou apagar a definição de um indicador	Alterar um indicador existente e validar que as alterações são correctamente efecutadas
US4	Como utilizador, consigo definir parâmetros para os indicadores	O sistema disponibiliza controlos para aplicar os filtros definidos
US5	Como utilizador, consigo definir relações entre indicadores. É possível navegar entre indicadores relacionados	O sistema disponibiliza controlos para navegar entre indicadores
US6	Como utilizador posso obter valores parametrizados para um indicador	Verificar que os resultados obtidos estão de acordo com os parâmetros utilizados
US7	Como utilizador, consigo ver indicadores relacionados (navegação horizontal)	Validar que o indicador relacionado dá a informação pretendida
US8	Como utilizador, consigo ver informação mais detalhada ou mais genérica de um indicador (navegação vertical)	Validar que o indicador relacionado dá a informação pretendida
US9	Como utilizador posso, partindo de um resultado de indicador, navegar para uma aplicação externa que dê mais informação sobre esse resultado	Os resultados dos indicadores mostram um url para outra aplicação, parametrizado com um identificador do resultado
US10	Os indicadores serão refrescados automaticamente segundo a sua definição de validade	Validar as datas de refrescamento do indicador

Tabela 1 – Requisitos funcionais

4.2 Requisitos de Qualidade

Os requisitos de qualidade serão descritos usando cenários de qualidade e seguindo a notação proposta em [3]. Esta notação, esquematizada na Figura 6, oferece uma descrição operacional do atributo e ao definir objectivos mensuráveis permite avaliar de forma concreta se os requisitos foram cumpridos. Um requisito de qualidade é então definido em 6 partes:

- **Fonte do estímulo**: uma entidade (utilizador, programador, sistema computacional, ou outro) responsável pela geração do estímulo;
- **Estímulo**: um evento que é recebido pelo sistema;
- **Estado**: um evento pode chegar em diferentes estados do sistema (em funcionamento, em sobrecarga, em falha, etc.);
- **Artefacto**: alvo do evento, pode ser o sistema todo ou uma parte deste;
- **Resposta**: actividade levada a cabo pelo sistema em resposta ao estímulo;
- **Medida da resposta**: medida que permite avaliar a resposta do sistema ao estímulo (latência, número de alterações, tempo da resposta, etc.);

Figura 6 - Cenário para descrição de um atributo de qualidade

O sistema proposto foi desenhado e construído de forma a exibir os atributos de qualidade enumerados e descritos na Tabela 2, apresentada abaixo:

Atributo	Fonte estímulo	Estímulo	Artefacto	Estado	Resposta	Medida da resposta
Modificabilidade	Programador	*Alterar a biblioteca de geração de gráficos*	Camada apresentação	Desenvolvimento	*Alteração de apenas uma classe*	Sem efeitos secundários
	Programador	*Mudar o repositório local da aplicação*	Camada lógica	Desenvolvimento	*Desenvolvimento de uma nova classe que respeite a interface definida para acesso ao repositório local*	Pequena alteração em apenas um componente
Personalização	Utilizador	*Modificar a disposição dos indicadores*	Camada apresentação	Operação normal	*O dashboard reflecte a alteração*	Imediato
	Utilizador	*Modificar a representação gráfica de um indicador*	Camada de apresentação	Operação normal	*A representação gráfica é alterada*	Imediato
Interoperabilidade	Programador	*Dotar a aplicação de capacidade de acesso a um novo tipo de repositório de dados*	Biblioteca de classes para acesso a dados	Operação normal	*Desenvolvimento de novas classes de acesso ao novo repositório*	Sem efeitos secundários
	Utilizador	*Consumir indicadores de uma nova base de dados*	Camada acesso a dados	Operação normal	*O sistema carrega automaticamente as classes necessárias para comunicar com o repositório de dados*	Sem efeitos secundários
	Utilizador	*Consumir indicadores de um novo web service*	Camada acesso a dados	Operação normal	*O sistema gera uma nova classe proxy para aceder aos métodos do novo web service*	Sem efeitos secundários
Extensibilidade	Programador	*Dotar a aplicação de capacidade de aplicar um novo tipo de filtro de pós-processamento*	Biblioteca de filtros de pós-processamento	Desenvolvimento	*Desenvolvimento de uma classe que implementa o novo tipo de filtro de pós-processamento*	Sem efeitos secundários
	Utilizador	*Aplicar um filtro de pós-processamento a um indicador*	Camada lógica	Operação normal	*O sistema passa a aplicar o filtro considerado ao indicador em questão*	Sem efeitos secundários
Desempenho	Utilizador	*Abrir dashboard*	Sistema	Operação normal	*Mostra Indicadores*	Latência ~1 segundo

Tabela 2 – Requisitos de qualidade

Capítulo 5

Desenho

A fase de desenho contemplou a análise dos objectivos, requisitos funcionais e de qualidade e, a partir destes, a elaboração de uma abordagem que permitisse criar uma solução que desse resposta ao problema proposto. Neste capítulo é descrito o trabalho realizado nesta fase, demonstradas as estratégias usadas para atingir os objectivos propostos, apresentando e detalhando os artefactos produzidos. Nas secções subsequentes serão apresentados os seguintes artefactos: **modelo da infra-estrutura**, **arquitectura de software inicial** e **modelo de dados**.

A infra-estrutura do sistema[4], ou estrutura de alocação, é um dos componentes da arquitectura de software. É um artefacto basilar no processo de desenho, no sentido em que dá uma visão macroscópica do sistema em análise. Consiste na decomposição da solução em peças de *software* tangíveis e na definição de protocolos ou mecanismos de comunicação entre estas. Os módulos e componentes identificados são elementos alocáveis a pessoas ou equipas de desenvolvimento durante a fase de construção e, posteriormente, alocáveis a unidades computacionais que devem cooperar e interagir no sentido de concretizar o sistema proposto. Considere-se como exemplo de uma infra-estrutura de sistema a seguinte descrição:

"O sistema é constituído por um componente servidor que será desenvolvido pela equipa A e será instalado no centro de dados localizado em Y e um componente cliente que será desenvolvido pela equipa B e executado através de browsers web. Os componentes interagem através do protocolo http."

[4] Normalmente denominado arquitectura do sistema. Neste documento será utilizado o termo infra-estrutura para evitar confusão com o conceito mais lato de arquitectura de software, descrito em [13]

Como pode ver-se através do exemplo supracitado, este artefacto só por si fornece pouca ou nenhuma informação sobre a implementação do sistema, não esclarece acerca das estruturas que o compõem, não permite inferir nem raciocinar sobre as características de qualidade que o sistema exibe e não refere as estratégias seguidas para atingir essas características. Convém salientar que o mesmo problema e a mesma infra-estrutura podem originar infinitas implementações distintas, todas elas com características e atributos concretos que são consequência directa das estratégias usadas para atingir a solução. Nesse sentido, o artefacto que descreve a infra-estrutura deverá ser complementado com informação sobre as outras estruturas da arquitectura de software no intuito de facultar uma visão mais profunda e detalhada do sistema.

A arquitectura de software de um programa ou sistema computacional consiste na estrutura ou estruturas do sistema, compostas por componentes de software, as relações entre eles e as suas propriedades externas visíveis[5].

A arquitectura de *software* é um artefacto do processo de desenho e define os elementos que constituem o sistema, quer sejam classes, módulos, processos, subsistemas, ou outros e as suas relações. Este artefacto fornece uma abstracção do sistema, no sentido em que a definição de um componente deve omitir os seus detalhes internos e focar-se nas características e propriedades perceptíveis para os outros componentes, por exemplo: definição de interfaces, protocolos de comunicação ou relações de hierarquia.

Do mesmo modo que o desenho de um edifício é constituído por diferentes plantas – planta dos diferentes pisos, planta eléctrica, planta do sistema de canalização, etc. – que dão informação sobre as diferentes perspectivas do edifício, uma arquitectura de software é constituída por diferentes estruturas ou vistas que definem os diferentes aspectos do sistema e documentam diferentes tipos de decisões arquitecturais, nomeadamente:

- Como deve o sistema ser estruturado a nível de unidades de código (**módulos**);
- Como deve o sistema ser estruturado num conjunto de elementos que tem um determinado comportamento em tempo de execução (**componentes**) e interacções (**conectores**);
- Como é que o sistema se vai relacionar com o ambiente externo e as estruturas físicas (máquinas, redes, pessoas).

[5] Adaptado de [13]

Estas decisões arquitecturais permitem agrupar as vistas de *software* em 3 tipos distintos:

1. **Estrutura de módulos**

Nesta estrutura o sistema é desenhado do ponto de vista do código e os elementos são módulos, ou unidades de implementação. Tipicamente, um módulo tem uma responsabilidade funcional bem definida e é dado pouco ênfase à forma como o software se manifesta em tempo de execução. Esta estrutura permite responder a perguntas como:

- *Quais as funcionalidades que cada módulo implementa?*
- *Para cada módulo, quais os módulos que podem ser acedidos directamente e quais os que têm acesso directo a este?*
- *Quais as relações de herança entre eles?*

2. **Estrutura de componentes e conectores**

Esta estrutura é definida à custa de componentes de execução (unidades de computação) e conectores (veículos de comunicação entre os componentes) e permite responder a perguntas como:

- *Quais são os principais componentes de execução?*
- *Quais os repositórios de dados partilhados?*
- *Qual é o fluxo de dados do sistema?*
- *Quais as partes do sistema que podem executar em paralelo?*

3. **Estrutura de alocação (infra-estrutura do sistema)**

Esta estrutura mostra a relação entre os elementos de computação e o ambiente externo onde estes elementos são desenvolvidos e executados e dá resposta às seguintes questões:

- *Em que processador é executado um determinado componente?*
- *Onde e como são armazenados os dados?*
- *Que equipas de desenvolvimento são responsáveis por implementar que partes do sistema?*

A arquitectura de *software* é um dos elementos mais importantes do processo de desenho e deve acompanhar e reflectir todas as decisões arquitecturais e de desenho que vão sendo tomadas durante a fase de desenvolvimento do sistema. Este artefacto permite, ainda na fase de desenho, discutir, criar ou reaproveitar estratégias que garantam determinados atributos de qualidade (padrões arquitecturais) e o facto de ser uma abstracção do sistema significa que é

reutilizável e pode ser aplicado para resolver problemas semelhantes. Uma vez que é composto por diferentes vistas, pode também ser utilizado como veículo de comunicação entre os vários interessados no sistema e dar a informação considerada relevante para cada um, como por exemplo:

- Informação de *time to market*[6], importante para os elementos da gestão da empresa;
- Informação de quais os componentes que apresentam maior risco, importante para a gestão do projecto;
- Informação de hierarquia de componentes, relevante para os elementos da equipa de desenvolvimento.

O modelo de dados de um sistema está directamente relacionado com o domínio ou contexto de aplicação e reflecte os conceitos subjacentes e as relações que estes estabelecem. A partir deste modelo deve ser possível perceber a estrutura, o formato dos dados e as regras de integridade que devem ser respeitadas. Este modelo poderá ser mapeado directamente na estrutura de tabelas de uma base de dados de suporte ao sistema.

5.1 Infra-Estrutura

A infra-estrutura do sistema, esquematiza na Figura 7, foi desenhada na fase inicial do projecto, após terem sido delineados os principais objectivos e requisitos da aplicação, consistindo numa solução de três camadas:

- Uma **camada de acesso a dados** que fornece uma abstracção das diferentes fontes de dados, que poderão encontrar-se em diferentes localizações e estar suportadas em diferentes tecnologias, a partir das quais serão extraídos os dados para os indicadores de desempenho.
- Uma **camada lógica** responsável por fazer a gestão dos indicadores e assegurar um mecanismo de *cache* com valores pré-calculados para os indicadores activos, delegando o cálculo desses valores aos respectivos repositórios de dados.
- Uma **camada de visualização**, que servirá de cliente da camada lógica, responsável pela interacção dos utilizadores com os indicadores de desempenho calculados.

[6] O *time to market* de um produto poderá ser optimizado utilizando componentes pré-fabricados (COTS), reaproveitando componentes de outros projectos ou estruturando o desenvolvimento numa ***Software Product Line*** ver [16].

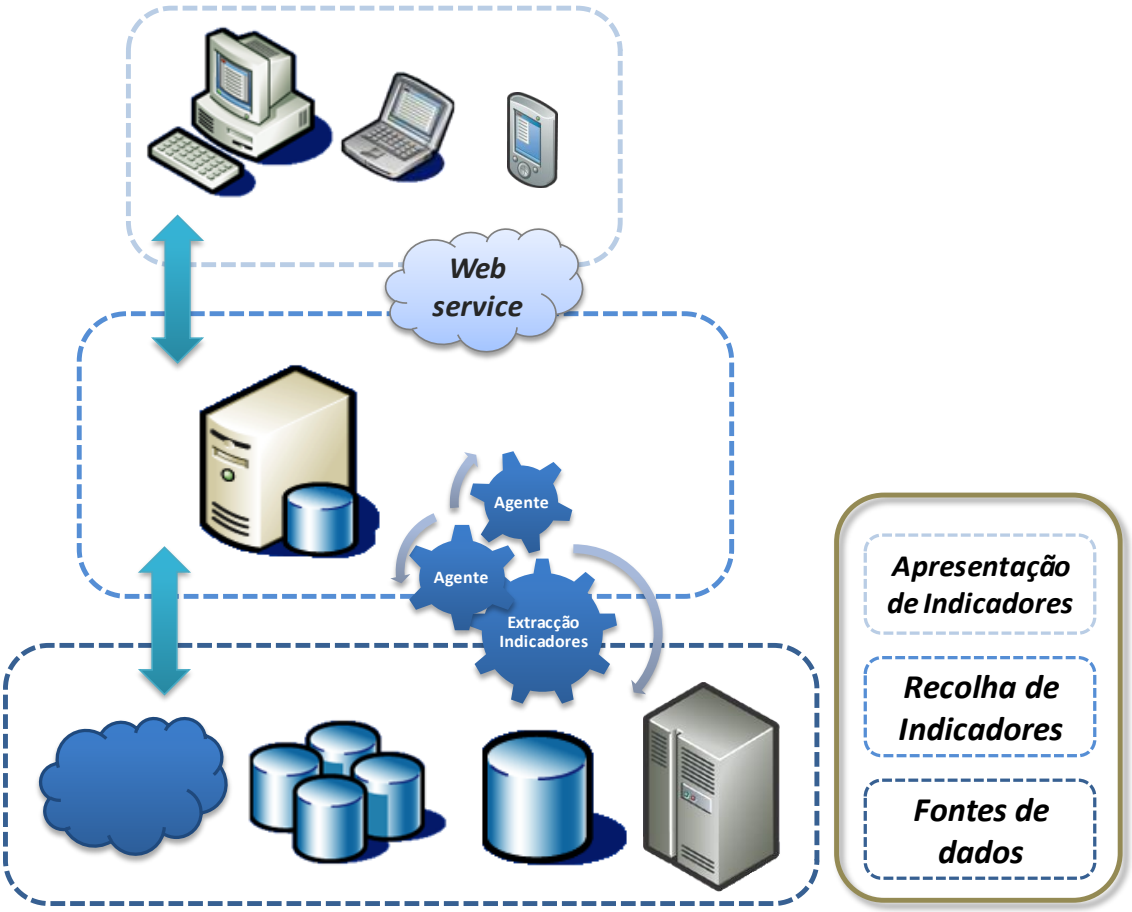


Figura 7 - Infra-estrutura do sistema

5.2 Arquitectura de Software

O desenho da arquitectura de software seguiu uma metodologia *Attribute Driven Design* e a sua primeira versão foi obtida logo após a primeira fase de levantamento de requisitos, tendo sido refinada ao longo da fase de construção e levando em linha de conta os seguintes factores:

- Princípios de *Low Coupling* e *High Cohesion*
- Requisitos de desempenho e escalabilidade
- Requisitos de modificabilidade e extensibilidade
- Requisitos de interoperabilidade

Esta versão inicial da arquitectura, ilustrada na Figura 8, demonstra já a utilização de algumas estratégias encapsuladas em padrões arquitecturais e tácticas, capazes de suportar os requisitos funcionais e atributos de qualidade identificados numa fase inicial. Nos subcapítulos seguintes enumeram-se as práticas mais relevantes seguidas na definição desta arquitectura de software.

A arquitectura final será apresentada e detalhada no Capítulo 7, que se debruça sobre a fase de construção.

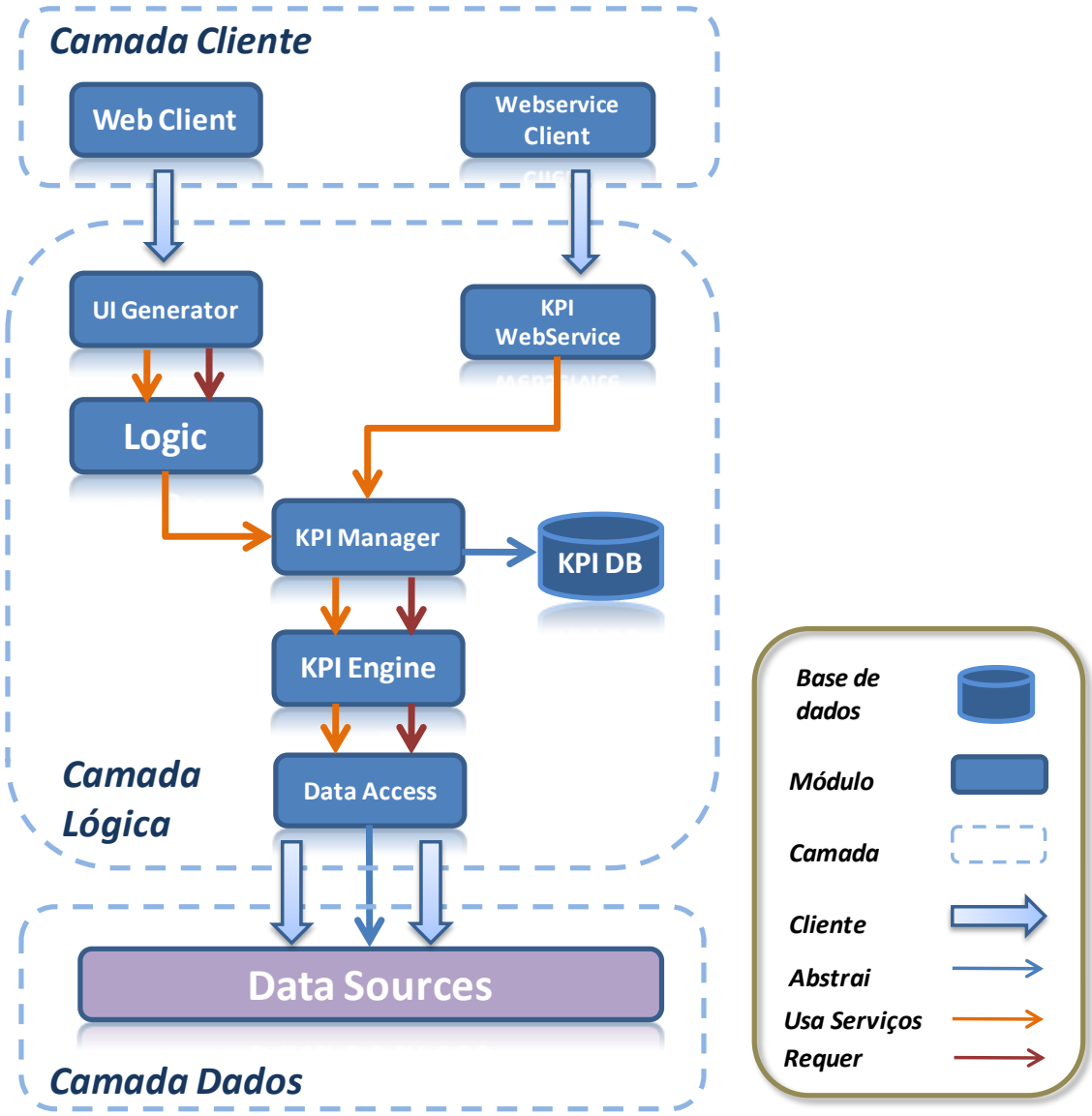


Figura 8 - Arquitectura de software inicial

5.2.1 Tácticas e Princípios

Os princípios chave que conduziram o processo de desenho da arquitectura foram os de *Low-Coupling* e *High-Cohesion*[7]. *Coupling* é uma métrica de qualidade de software que se refere à dependência entre diferentes módulos e ocorre quando um módulo depende das especificidades de outro para

[7] Estas métricas foram inventadas por Larry Constantine e foram pela primeira vez publicadas em [15]

desempenhar as suas funções, como por ex., acesso directo a dados locais ou partilha de variáveis globais. Isto implica que se houver alguma alteração num elemento partilhado, esta alteração repercutir-se-á em todos os componentes que dependem deste. É desejável manter um baixo nível de dependência entre módulos, no sentido de prevenir o *efeito dominó*[8] quando é efectuada alguma alteração.

Cohesion é outra medida de qualidade importante e refere-se ao grau em que um módulo ou componente agrega um conjunto de funcionalidades que contribuem para uma responsabilidade comum, por exemplo, um sistema financeiro composto por um módulo que agrega todas funções de cálculo e outro que agrupa todas as funcionalidades de geração de relatórios. Ter módulos com responsabilidades bem definidas contribui para um sistema mais fácil de compreender, testar, reutilizar, manter e estender.

O primeiro padrão arquitectural considerado foi o *Multi-Tier* (também conhecido por *N-Tiers*), este padrão define uma arquitectura de cliente-servidor com várias camadas (*tiers*) em que cada qual tem uma responsabilidade clara, contribuindo para o fluxo de dados e objectivos do sistema. Cada camada é cliente da camada imediatamente abaixo e servidor da imediatamente acima e comunica com estas através de interfaces e/ou protocolos bem definidos. Esta característica permite obter sistemas com alta interoperabilidade e extensibilidade, no sentido em que é possível, desde que sejam respeitadas as interfaces ou protocolos de comunicação definidos, criar novos componentes que comunicam com as camadas existentes ou até mesmo substituir qualquer das camadas sem afectar o funcionamento do sistema e contribuindo para a escalabilidade, uma vez que origina um sistema descentralizado. Estas propriedades, como em qualquer sistema distribuído, são atingidas à custa de um maior consumo de recursos e de um menor desempenho devido aos níveis de indirecção que cada camada introduz.

Este padrão arquitectural contribui fortemente para os princípios de *low-coupling* (os componentes interagem através de pontos de comunicação bem definidos e não precisam de conhecer os detalhes de implementação dos outros) e *high-cohesion* (cada camada tem um conjunto de responsabilidades circunscrito).

[8] Do inglês *Ripple Effect.* Verifica-se quando uma alteração num ponto do sistema afecta outras partes do mesmo, propiciando comportamentos inesperados e/ou obrigando a alterações extensas.

Para construir um sistema que cumpre com os requisitos de modificabilidade e extensibilidade propostos, foram identificados *a priori* os pontos de variabilidade ou de mudança e, posteriormente, consideradas algumas tácticas relevantes em tempo de desenvolvimento e em tempo de execução, que contribuem para a obtenção destes atributos, nomeadamente:

- ***Information Hiding* e coerência semântica***: Information Hiding*[9], consiste em "esconder" detalhes ou implementações concretas de módulos ou componentes, que são susceptíveis de ser modificados, por detrás de interfaces que se mantêm imutáveis. Desta forma, se ocorrer uma alteração na implementação de um módulo ou componente e a semântica da interface não for alterada (coerência semântica) os módulos que dependem deste não serão afectados. Esta é uma das técnicas que ajuda a atingir *low-coupling.*
- **Abstracção de serviços comuns**: Criação de módulos especializados que abstraem serviços usados por vários componentes como, por exemplo, acesso a bases de dados ou interacção com dispositivos externos. Esta técnica, além de melhorar a *cohesion* da solução e promover a reutilização de código, contribui para a modificabilidade do sistema, no sentido em que se ocorrer uma alteração em algum destes serviços terão de ser feitas modificações apenas no componente os expõe.
- **Arquitectura *Plug&Play***: Arquitectura que permite o carregamento de componentes em tempo de execução, de forma a estender ou modificar o comportamento e as funcionalidades do sistema, carregando, configurando ou retirando componentes.

5.2.2 Desenho da Arquitectura

O primeiro atributo considerado no desenho da arquitectura do sistema aqui apresentado foi o **desempenho** que pode ser medido, neste caso concreto, através da latência percepcionada pelos clientes no acesso aos indicadores. De forma a construir um sistema com um bom desempenho foi implementado um mecanismo de *cache* que armazena valores pré-calculados para os indicadores definidos, garantindo que quando essa informação é solicitada não tem de ser recolhida no repositório original, podendo assim ser prontamente disponibilizada.

Em seguida foi contemplada a **interoperabilidade** do sistema, ponderada sob três vectores: 1) capacidade do sistema consumir dados de repositórios

[9] Conceito publicado pela primeira vez em 1972 por David Parnas em [18]

suportados em tecnologias distintas; 2) capacidade de disponibilizar a informação dos indicadores de negócio sob diferentes formas e 3) capacidade de interligação transversal entre os dados dos indicadores disponibilizados.

Para este feito, foi considerado o padrão arquitectural *N-tiers* tendo sido definidas as seguintes camadas:

- **Camada de acesso a dados** – abstrai o processo de obtenção de dados a partir dos repositórios originais. Esta camada é acessível através de um conjunto de protocolos: protocolos de acesso e comunicação com bases de dados e protocolos para aceder a dados expostos através de *webservices*;
- **Camada de negócio** – tem a responsabilidade de orquestrar o processo de obtenção de dados, construção da *cache* de valores para os indicadores de negócio e gestão destes indicadores, cujo acesso é assegurado através do protocolo http;
- **Camada cliente** – suporta a parametrização, visualização e exploração por parte do utilizador final, dos dados de desempenho apresentados sob a forma de indicadores.

A aplicação deste padrão arquitectural garante e promove os atributos de interoperabilidade e escalabilidade.

No que concerne aos atributos de **modificabilidade** e **extensibilidade** foram identificados os seguintes pontos de variabilidade como os mais relevantes:

- Tecnologia subjacente aos repositórios origem dos dados;
- Tecnologia de bases dados para o repositório local;
- Forma de visualização dos indicadores;
- Filtros de pós-processamento de indicadores.

Atentando aos objectivos de **modificabilidade** e **extensibilidade** supracitados, o recurso às técnicas de ***information-hiding*** e **coerência semântica** determinou o desenvolvimento de uma interface que "mascara" os detalhes de acesso às diferentes fontes de dados (*DataAccess*) e a criação de um componente (*Logic*) na camada de visualização, responsável por formatar os dados dos indicadores e gerar as representações gráficas adequadas.

A técnica de **abstracção de serviços comuns** foi concretizada através de um componente que abstrai as funcionalidades de acesso ao repositório de dados local (*KPI Manager*).

Finalmente, foi empregue a técnica de *Plug&Play* para habilitar o sistema da capacidade de injectar, em tempo de execução, novos componentes aptos para a

extracção de indicadores a partir de fontes de dados não previstas em tempo de desenvolvimento e ainda para permitir a inclusão/remoção de filtros de pós-processamento de indicadores.

5.3 Modelo de dados

O modelo de dados que suporta a definição de indicadores e persistência local dos dados dos indicadores de desempenho, foi desenhado seguindo o modelo relacional e está esquematizado na Figura 9. A Tabela 3 dá informação descritiva das tabelas e dos vários campos que as constituem.

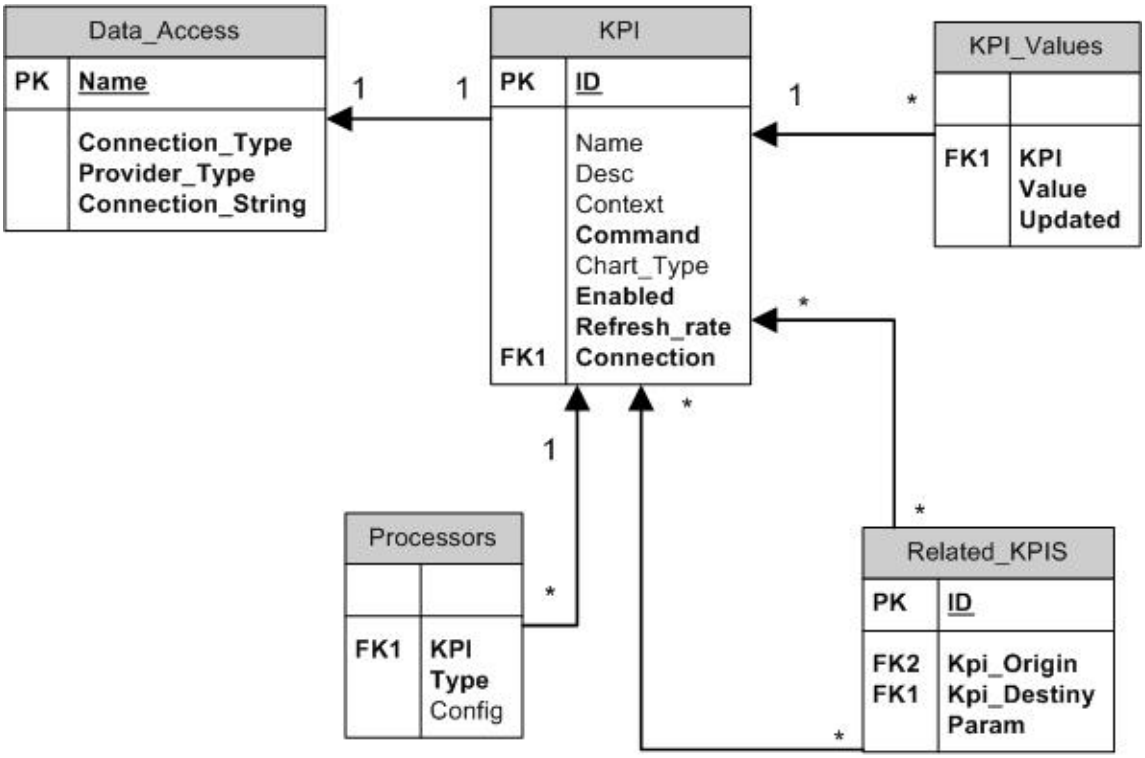


Figura 9 - Modelo de dados

Tabela	Campo	Descrição
KPI	ID	Identificador
	Name	Designação
	Desc	Descrição
	Context	Contexto (permite agrupar conjuntos de indicadores)
	Command	Operação a executar: query sql ou método *web service*
	Chart_Type	Tipo de gráfico
	Enabled	Indicador Activo?
	Refresh_Rate	Taxa de refrescamento do indicador de em minutos
	Connection	Nome da ligação ao repositório de dados
Data_Access	Name	Nome da ligação ao repositório de dados
	Connection_Type	Tipo de acesso: Web service ou Base de dados
	Provider_Type	Tipo de dados (Classe) que concretiza o acesso
	Connection_String	String que configura o acesso à fonte de dados
KPI_Values	KPI	Identificador do indicador
	Value	Valor
	Updated	Data em que o valor foi calculado
Related_KPIS	ID	Identificador
	KPI_Origin	Indicador origem da relação
	KPI_Destiny	Indicador destino da relação
	Param	Parâmetro de entrada no indicador destino
Processors	KPI	Identificador do indicador
	Type	Tipo de dados (Classe) que concretiza o filtro de pós-processamento
	Config	Nome da secção de configuração do indicador

Tabela 3 – Descrição do modelo de dados

Capítulo 6

Contexto e Enquadramento Tecnológico

Este capítulo pretende apresentar o contexto e enquadramento tecnológico em que a aplicação foi desenvolvida, enumerando e descrevendo as tecnologias e ferramentas utilizadas. Na primeira parte serão apresentadas as ferramentas que constituem o ambiente de desenvolvimento utilizado e suportam o processo descrito no Capítulo 3. Na segunda parte serão detalhadas as tecnologias escolhidas e como estas se relacionam no sentido de concretizar o sistema proposto. As figuras 10 e 11 esquematizam respectivamente, as pilhas tecnológicas usadas para o processo de desenvolvimento e o sistema desenvolvido.

6.1 Processo de desenvolvimento

A construção do sistema proposto foi baseada no processo de desenvolvimento pormenorizado no Capítulo 3 e, para esse efeito, foi criado um ambiente de desenvolvimento com recurso a um conjunto de ferramentas que permitiram atingir os objectivos propostos a este nível. A lista abaixo e a Figura 10 descrevem as ferramentas utilizadas para a construção do sistema e para a garantia da qualidade do software desenvolvido:

- ***ScrumWorks*** – Ferramenta de *Scrum* utilizada para gestão do projecto;
- ***Visual Studio 2008*** – *IDE* da Microsoft para desenvolvimento de aplicações na *framework* Microsoft.NET;
- ***nAnt*** – Ferramenta *open-source* para automatização do processo de construção;
- ***CruiseControl .NET*** – Servidor de construção e *framework* para implementação e controlo do processo de integração contínua;
- ***nUnit*** – *Framework open-source* para implementação de testes unitários;

- ***nMock*** – Biblioteca de objectos *Mock*[10] para a *framework .NET;*
- ***nCover*** – Ferramenta que analisa a cobertura de código atingida por uma bateria de testes;
- ***Watin*** – *Framework* que permite automatizar acções em browsers *web*, facilitando assim a criação de casos de teste para interfaces *web* que podem ser repetidos, automatizados e integrados com a *framework* de testes unitários ***nUnit***;
- ***FxCop*** – Ferramenta de análise estática de código que permite efectuar a sua validação quanto a erros comuns, falhas de segurança, desempenho e correcção, entre outros.

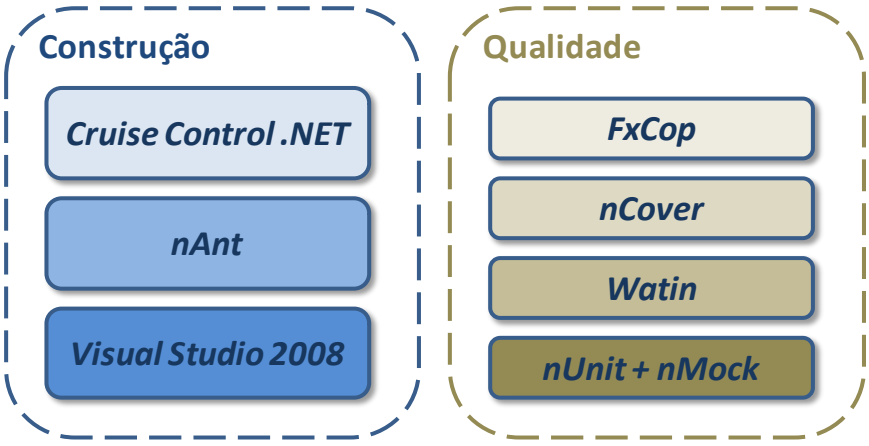


Figura 10 - Pilha tecnológica (ambiente de desenvolvimento)

6.2 Produto

Com o amadurecimento das redes de computadores e de tecnologias que permitem gerar conteúdo dinâmico, através da execução de código aplicacional em servidores *web*, passou a ser possível construir aplicações que executam em servidores e apresentam os resultados gerando páginas *HTML*, que podem ser visualizadas através de qualquer *browser*. Este tipo de aplicações têm vantagens significativas quando equiparadas a aplicações tradicionais que são executadas localmente, como: a facilidade de instalação e actualização, configuração centralizada no servidor e economia de recursos. Apesar deste modelo de computação apresentar algumas desvantagens, por ter um desempenho limitado, dependência da disponibilidade da rede e dificuldade de criação de interfaces mais complexas, para alguns tipos de aplicações é indubitavelmente a

[10] *Mocks* são objectos que simulam dependências de outros objectos ou dispositivos, com o objectivo de tornar os testes mais objectivos e independentes

melhor opção. Tendo estas duas alternativas em mente e considerando a natureza das aplicações a desenvolver, bem como as expectativas do mercado em relação a este tipo de ferramentas, optou-se pelo desenvolvimento de aplicações suportadas em ambiente *web*.

No que concerne ao desenvolvimento quer da plataforma, quer das aplicações cliente, foram empregues as tecnologias e ferramentas de desenvolvimento *Microsoft*; a lista abaixo descreve estas tecnologias e a Figura 11 esquematiza a forma como estas se relacionam:

- ***Microsoft .NET Framework 3.5*** – Plataforma *Microsoft* para desenvolvimento e execução de aplicações;
- ***SQLite*** – SGBD relacional que permite criar bases de dados embebidas nas aplicações, evitando assim a necessidade de instalação e administração destas bases de dados e a necessidade de comunicação inter-processos entre o repositório de dados e a aplicação. Características que a tornam adequadas para o papel de repositório local de dados e de configurações;
- ***Microsoft ASP .NET Webservices*** – Tecnologia *Microsoft* para criação e execução de *webservices;*
- ***Microsoft ASP.NET 2.0*** – Tecnologia *Microsoft* para criação de sítios *web* com páginas dinâmicas e aplicações *web;*
- ***Microsoft Windows SharePoint Services 3.0*** – O *Windows Sharepoint Services* é uma versão gratuita do *Microsoft Office Sharepoint Server*, oferecendo as funcionalidades base de criação de portais corporativos e intranets colaborativas, aplicações *web* de gestão documental, gestão de conteúdos e automatização de processos de negócio;
- **Microsoft *.NET Compact Framework*** – Versão reduzida da plataforma *.NET Framework* que permite criar aplicações para o sistema operativo *Windows Mobile* que podem ser executadas em dispositivos móveis.

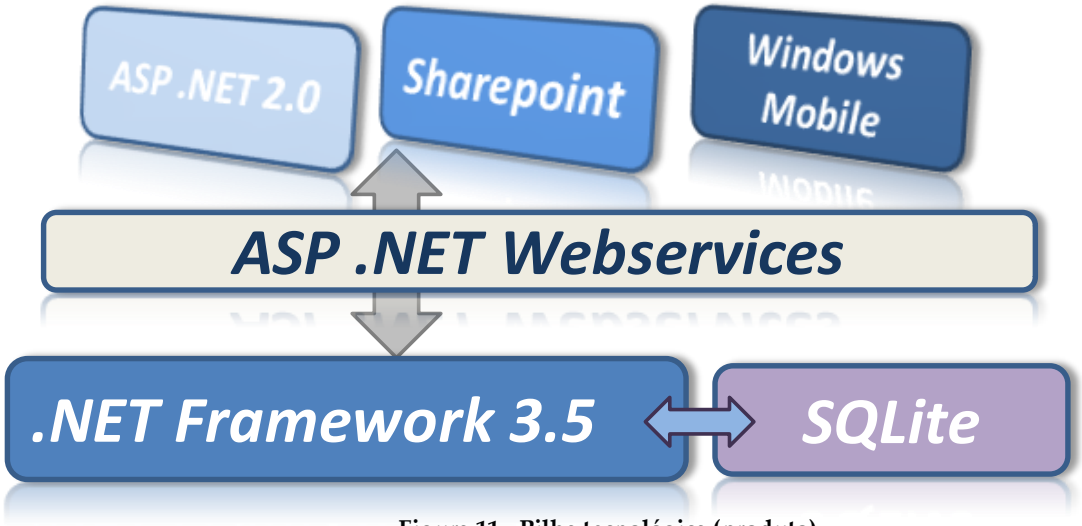


Figura 11 - Pilha tecnológica (produto)

Capítulo 7

Construção

Na fase de desenho inicial, foram analisados e levantado*s os requ*isitos de qualidade que a ferramenta deveria exibir, quais as principais funcionalidades suportadas (Capítulo 4), quais os princípios que deveriam conduzir o processo de desenho e a partir destes elementos foi criado um primeiro esboço da arquitectura de software (Capítulo 5). Durante a fase de construção, a arquitectura proposta foi refinada como reflexo de uma melhor compreensão dos requisitos, dos desafios apresentados e das restrições e limitações impostas pela escolha das tecnologias.

Este capítulo pretende revisitar a arquitectura de software proposta na fase de desenho inicial, explicar o fluxo de informação do sistema ***TrueKPI***, descrever, de uma perspectiva técnica, como foi construída a ferramenta e de que forma esta implementação dá resposta a todos os requisitos funcionais e não funcionais propostos.

O fluxo de dados principal da aplicação é constituído por 3 pontos-chave: **cálculo** dos indicadores de desempenho a partir dos repositórios de dados de negócio, **armazenamento** desses indicadores e construção da *cache* de valores e, finalmente, a **disponibilização** dessa informação. Nas secções seguintes serão detalhados estes pontos, os componentes desenvolvidos para cada uma destas fases, as suas respectivas funções e de que forma cooperam para atingir os objectivos propostos tanto a nível de funcionalidades como a nível de propriedades não funcionais da aplicação.

7.1 Regresso à arquitectura de software

"Um trabalho nunca está acabado exceptuando em caso de algum acidente como a fadiga, satisfação, necessidade de entrega ou morte…"

Paul Valéry

A fase de construção da aplicação iniciou-se com uma reavaliação dos requisitos propostos e subsequente evolução e reengenharia da arquitectura de software existente, motivada pela identificação de alguns pontos de optimização e por uma melhor compreensão e definição dos objectivos do sistema. As alterações propostas, elencadas abaixo, foram concebidas tendo sempre em mente os princípios de desenho, objectivos funcionais e de qualidade já conhecidos e acomodados.

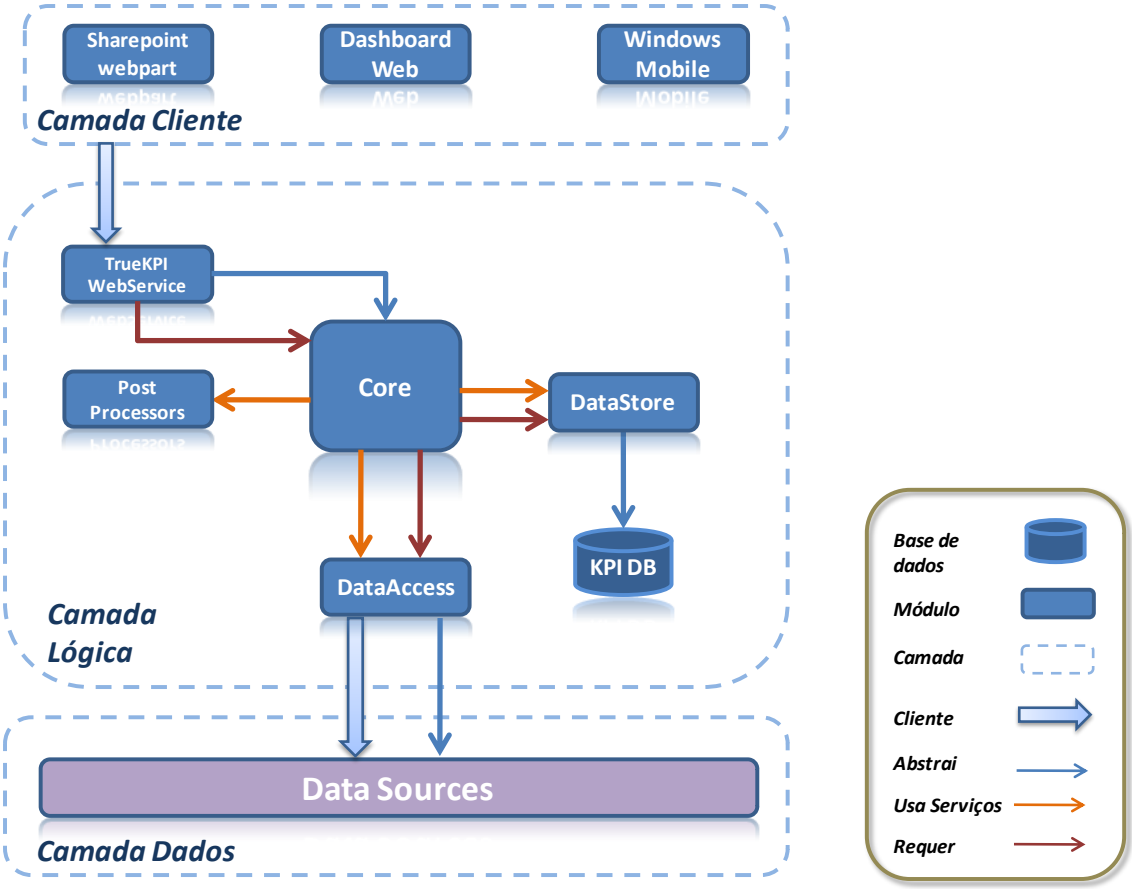


Figura 12 - Arquitectura de software revisitada

7.1.1 Especialização dos componentes de acesso a dados

Uma das vertentes da interoperabilidade pretendida para o sistema desenvolvido está relacionada com a aptidão deste para extrair indicadores de diferentes tipos de repositórios de dados. Apesar de se pretender que o acesso a dados seja suficientemente genérico para suportar diferentes tecnologias de armazenamento e de disponibilização de dados, é necessário criar componentes capazes de lidar com as particularidades de cada tipo de acesso. Para esse efeito, foram identificados dois componentes abstractos (que foram, *a posteriori*, instanciados em componentes concretos que comunicam com fontes de dados concretas). Um com a capacidade de aceder a informação residente em bases de dados e outro que permite consultar dados expostos através de *web services* (*Database DataAccess* e *Webservice DataAccess* respectivamente).

7.1.2 Módulo de pós-processamento de indicadores

No que concerne à extensibilidade do sistema, uma das estratégias adoptadas consistiu em dotar o sistema da capacidade de injecção e remoção de filtros de pós-processamento com o propósito de enriquecer os resultados dos indicadores obtidos. Com a criação do módulo *PostProcessors* foram discernidas as responsabilidades de encontrar e carregar, em tempo de execução, os filtros de pós-processamento pretendidos e de aplicar estes filtros aos indicadores correctos.

7.1.3 Padrão arquitectural *Pipe&Filters*

A funcionalidade de pós-processamento de resultados dos indicadores foi implementada utilizando o padrão arquitectural *Pipe&Filters* (ver [4] e Figura 13). Este padrão utiliza como **componentes** o conceito de filtros (*filters*) e como **conectores** o conceito de canais (*pipes*). Um filtro pode ser visto como uma "caixa preta" que recebe, no canal de entrada, uma mensagem ou fluxo de dados, realiza alguma transformação sobre estes e devolve o resultado num canal de saída, que por sua vez encaminha a mensagem ao próximo filtro. Estes filtros não têm estado, conhecimento do contexto em que estão inseridos, nem da forma como os dados são recebidos e enviados. Este padrão arquitectural promove a fiabilidade, uma vez que os filtros são componentes independentes e com âmbito bem circunscrito (*high-cohesion*) e a reutilização, no sentido em que os filtros não têm estado nem fazem assumpções sobre o seu contexto de execução. Estes factores, aliados a uma comunicação feita através de troca de mensagens garante os mais baixos níveis de *coupling*. Como contrapartida,

sistemas muito complexos estruturados segundo este padrão exigem um maior esforço de manutenção, são mais difíceis de depurar, e apresentam a desvantagem de dificultar o tratamento de falhas, uma vez que os filtros não tendo conhecimento do contexto global em que estão inseridos, encontram-se pela sua natureza, limitados na tomada de medidas apropriadas em caso de erro.

Figura 13 - Arquitectura *Pipe&Filters*

7.1.4 Componente *Worker*

Foi considerada a refactorização do componente *Engine,* ficando este responsável por orquestrar o processo de obtenção dos vários indicadores e delegando a extracção de dados no componente *Worker*. Este componente sendo responsável pela extracção de cada indicador, tem o ónus de contactar o repositório de dados adequado e calcular o respectivo indicador de desempenho. Tem ainda o encargo de, caso seja necessário, aplicar os filtros de pós-processamento correctos. Esta funcionalidade é atingida através do uso do padrão arquitectural *Pipe&Filters,* sendo que os filtros são carregados, dinamicamente para uma lista e aplicados sequencialmente. Ou seja, os canais de comunicação são concretizados através do mecanismo de invocação explícita, passando a mensagem (neste caso o resultado do indicador) como parâmetro dessa invocação.

7.1.5 Acesso ao repositório local

Reiterando os princípios de separação de responsabilidades e abstracção de serviços comuns, foram removidas do componente *Manager* as tarefas de acesso e gestão da base de dados local, para as quais foi contemplada a criação de um módulo (*DataStore)* que abstrai e encapsula todos os serviços de interacção com este repositório.

7.1.6 Ponto único de acesso à camada lógica

Foi uniformizada a forma de acesso à camada lógica, no sentido de simplificar a arquitectura e alcançar uma separação clara a nível de responsabilidades e consequentemente reduzir a dependência desta camada

com a de visualização, permitindo assim que ambas possam evoluir de forma independente e sem efeitos colaterais noutras partes do sistema. Para este efeito, o ponto de acesso à camada lógica foi concretizado através de um *web service* que serve de mediador para todas as interacções com esta camada.

7.1.7 Redefinição da camada visualização

A outra faceta da interoperabilidade considerada, prende-se com a possibilidade de poder visualizar a informação dos indicadores de negócio de formas distintas. Nesse sentido e após a uniformização do acesso à camada lógica considerou-se a criação de uma camada de visualização composta por diferentes clientes: um *dashboard* Web, um componente *webpart* que representa um indicador e que pode ser aplicado a um portal *Sharepoint* e um cliente rico que executa em ambiente *Windows Mobile* (ver secção 7.6

A evolução proposta para a arquitectura de *software* promoveu a modificabilidade, extensibilidade, interoperabilidade e os princípios de *low-coupling e high-cohesion* capitalizando assim no sistema obtido, os benefícios destas qualidades discutidos no capítulo do desenho (Capítulo 5).

7.2 Padrões de desenho e arquitectura definida

"Cada padrão descreve um problema que ocorre vezes sem conta no nosso contexto e descreve a base para a solução desse problema, de forma a que esta solução possa ser aplicada milhões de vezes e sem que seja usada da mesma forma duas vezes…"

Christopher Alexander

O conceito de padrões de desenho foi adoptado a partir do trabalho do arquitecto Christopher Alexander, explanado em [5], para diversas áreas de estudo incluindo a engenharia de *software*. Nesta, um padrão de desenho pode ser visto como *uma descrição de classes e objectos que comunicam entre si, especializados para resolver um problema de desenho genérico num contexto particular*[11]. Os padrões de desenho não definem técnicas de programação e não podem ser directamente transformados em código, são antes tácticas empíricas que provaram ser adequadas para resolver determinados problemas recorrentes

[11] Adaptado de [19]

de desenho. No decorrer da fase de implementação foram aplicados os seguintes padrões de desenho[12]:

Padrão de desenho	Classes Envolvidas
Abstract Factory	DataStoreFactory
	IValuesManager
	XMLValuesManager
	DynamicDataValuesManager
	IDataStore
	SQLiteDataStore
Dependency Injection	IDataStore
	Worker
	Engine
Facade	TrueKPI Webservice
	Manager
Factory Method	DataAccessFactory
	IDataAccess
	DataBaseAccess
	WebserviceAccess
	PostProcessorFactory
	IPostProcessor
	PostProcessor
Proxy	TrueKPI Webservice Proxy
Singleton	Manager
Strategy	IValuesManager
	DynamicDataValuesManager
	XMLValuesManager

Tabela 4 – Padrões de desenho aplicados

A aplicação dos padrões de desenho supracitados, bem como as alterações descritas no início do capítulo, potenciaram a evolução da arquitectura de software do sistema. A Figura 14 apresenta, respectivamente, os esquemas dos componentes principais e elementos auxiliares, que constituem a arquitectura final para a camada lógica.

[12] Ver anexo a explicação dos padrões de desenho e a forma como estes contribuíram para a arquitectura de *software*

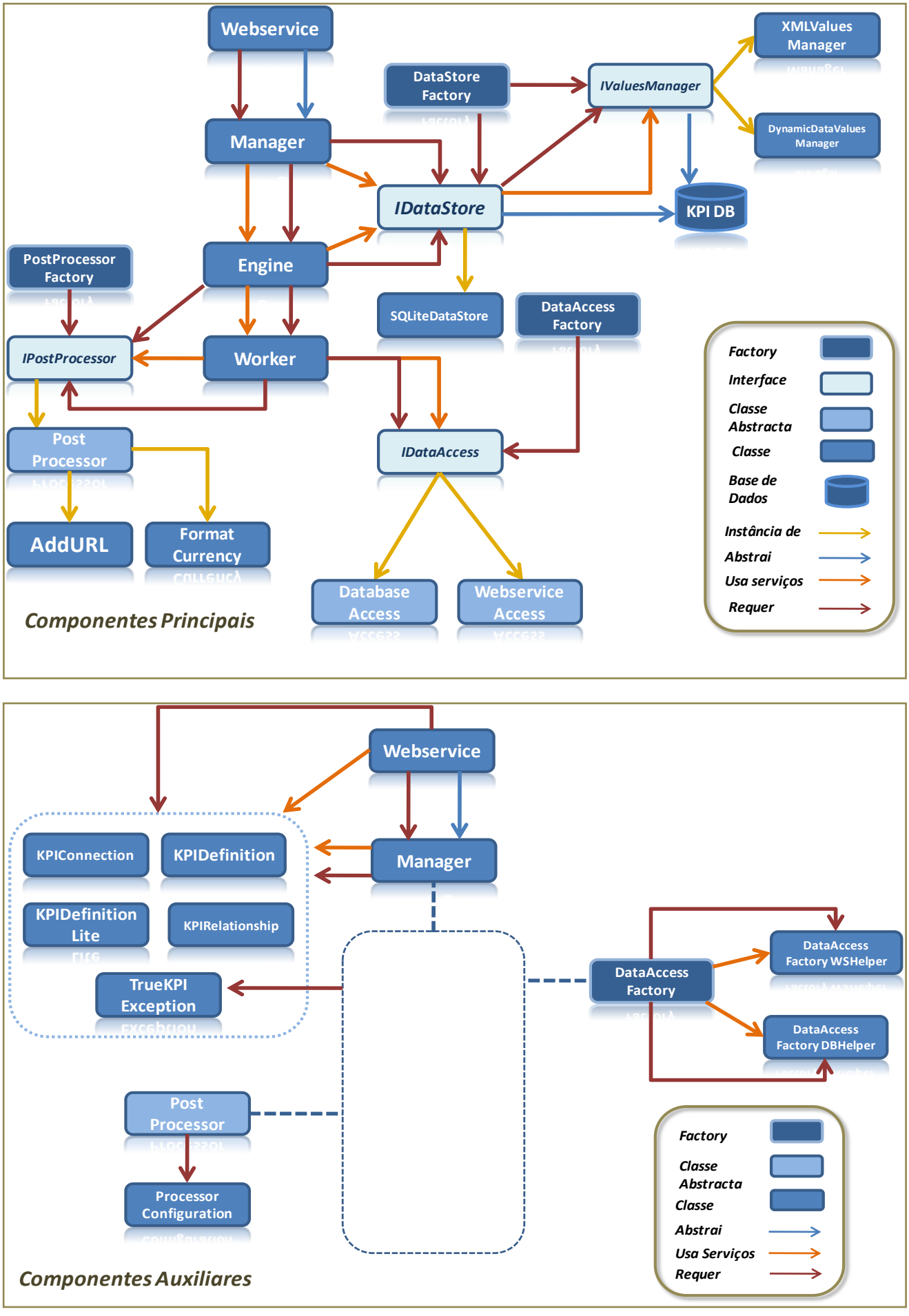


Figura 14 - Arquitectura camada lógica

7.3 Criar e Gerir Indicadores

O primeiro passo para a utilização da aplicação ***TrueKPI,*** após a sua instalação, consiste em configurar e parametrizar o sistema de acordo com contexto em causa. Este passo inclui a definição de configurações de acesso a origens de dados, de indicadores de desempenho, de relações entre indicadores e, sempre que aplicável, da atribuição de filtros de pós-processamento aos indicadores. As operações de configuração e parametrização do sistema são asseguradas pelo componente *Manager* e a informação será guardada na base de dados local do sistema, invocando os métodos disponibilizados pelo componente *IDataStore* para acesso a este repositório.

Na lista abaixo estão sumariadas as operações, disponíveis para utilizadores com o perfil de administrador de sistema, que o sistema suporta a nível de criação e gestão de indicadores.

1. **Definições de acessos a fontes de dados**
    a. Criação
    b. Edição
    c. Eliminação
2. **Indicadores de desempenho**
    a. Criação
    b. Edição
    c. Eliminação
    d. Definição de parâmetros
3. **Relações entre indicadores**
    a. Criação
    b. Eliminação
4. **Filtros de pós-processamento**
    a. Atribuição
    b. Dissociação

7.3.1 Linguagem de definição de comandos

Um dos requisitos funcionais identificados consiste na possibilidade de criar indicadores que possam ser parametrizados por um utilizador final, ainda que com um perfil avançado de forma a obter resultados mais específicos, como por exemplo, num indicador que mostra os totais de vendas de determinado produto, ser possível filtrar os resultados para mostrar apenas os valores de venda para um determinado mês ou para um determinado vendedor.

A primeira abordagem considerada para implementar este requisito consistiu na definição de filtros que seriam armazenados na base de dados local e seriam concatenados à expressão base, quando invocados, de forma a obter os resultados específicos. Esta abordagem apresenta algumas desvantagens, nomeadamente:

- A definição dos indicadores é menos intuitiva;
- A manutenção dos indicadores é mais complexa;
- É menos genérica, uma vez que os filtros são concatenados à expressão original.

Optou-se então por definir uma pseudo-linguagem, que permita introduzir filtros directamente na expressão do indicador. Esta linguagem deve ser suficientemente expressiva e genérica para permitir introduzir filtros em qualquer ponto do comando e ao mesmo tempo suficientemente intuitiva para permitir que as expressões obtidas sejam de fácil compreensão e manutenção.

Para definir um indicador parametrizável calculado numa base de dados, basta escrever a *query SQL*, identificando com referências os locais onde serão aplicados os filtros. Os comandos deverão ser escritos num dos seguintes formatos:

- ***<expressão sql> [campo a filtrar : nome da variável] </expressão sql>***
- ***<expressão sql> [expressão por omissão : nome da variável] </expressão sql>***

Como exemplo, apresenta-se a seguinte expressão:

```
select cta.ctaNomeEntidade entidade, ele.doevalor divida
from aifdocelemento ele
join aifdoccontabilistico doc on ele.dcoid = doc.dcoid
join aifconta cta on cta.ctaId = doc.ctaId
where ele.doeconceito = 197
and [ele.doevalor:valor1]
and [ele.doevalor > 50000:valor2]
order by ele.doevalor;
```

Esta linguagem permite ainda definir indicadores calculados à custa de invocação de operações em *web services*, sendo necessário indicar, neste caso, o nome da operação, os parâmetros e os respectivos tipos[13]. Para especificar os tipos dos parâmetros, estes devem ser identificados de acordo com o formato abaixo descrito.

<nomeOperação>(*[parametro1][,…,][parametroN])* </nomeOperação>*;*

[13] É necessário conhecer o tipo dos parâmetros de forma a poder distinguir a operação a invocar, no caso de existirem várias operações com o mesmo nome, mas com tipos de parâmetros diferentes

Os indicadores extraídos a partir de *webservices* também podem ser parametrizáveis e os filtros são definidos no seguinte formato:

<operação> [valor por omissão : nome da variável] </operação>

Considere-se, a título de exemplo, a definição da expressão de indicador representada abaixo:

```
getInvoices("DeptA",100,[50:valorMáximo],<10-10-2008>);
```

A pseudo-linguagem usada para a construção dos comandos que concretizam os indicadores permite definir tanto indicadores calculados em bases de dados, como em *web services*, incluindo a parametrização de valores por omissão e, para o caso de indicadores calculados em bases de dados, parâmetros opcionais.

7.4 Obter Indicadores

O fluxo de disponibilização de indicadores de desempenho, começa com a extracção dos valores dos indicadores a partir dos sistemas de gestão de bases de dados ou outras fontes de dados que dão suporte ao negócio. O sistema ***TrueKPI*** foi desenhado de forma a poder vir a permitir a extracção de dados de fontes suportadas em tecnologias distintas e de adicionar, *a posteriori*, novas fontes de dados não previstas inicialmente, usando uma técnica conhecida como *late binding* ou *lazy binding* que consiste em diferir até à altura de execução a escolha dos componentes que, neste caso, concretizam as comunicações com os repositórios de dados. Nas subsecções seguintes serão detalhados os mecanismos usados para materializar os acessos aos repositórios de dados, os componentes intervenientes na extracção de indicadores e na aplicação dos filtros de pós-processamento, bem como as técnicas usadas para calcular indicadores parametrizados e indicadores relacionados.

7.4.1 Acesso a dados

Como foi referido previamente, o sistema apresentado tem a capacidade de consumir informação a partir de diversas fontes de dados, em particular, a partir de bases de dados e de *web services*. Estas duas tecnologias têm particularidades muito distintas e a comunicação com cada uma apresenta desafios que foram resolvidos com diferentes abordagens.

Bases de dados

A comunicação e interacção com bases de dados para extracção de informação foi implementada utilizando a infra-estrutura disponibilizada pela *framework .NET*, para comunicação com bases de dados – a *ADO .NET* –. Esta infra-estrutura oferece duas interfaces que foram utilizadas no sentido de atingir os objectivos propostos, concretamente:

- ***IDdConnection***: esta interface define os métodos de ligação e comunicação com a fonte de dados;
- ***IDbCommand***: interface que oferece os métodos de execução para comandos de *Data Definition Language* e *Data Modification Language* válidos no contexto de cada fonte de dados concreta.

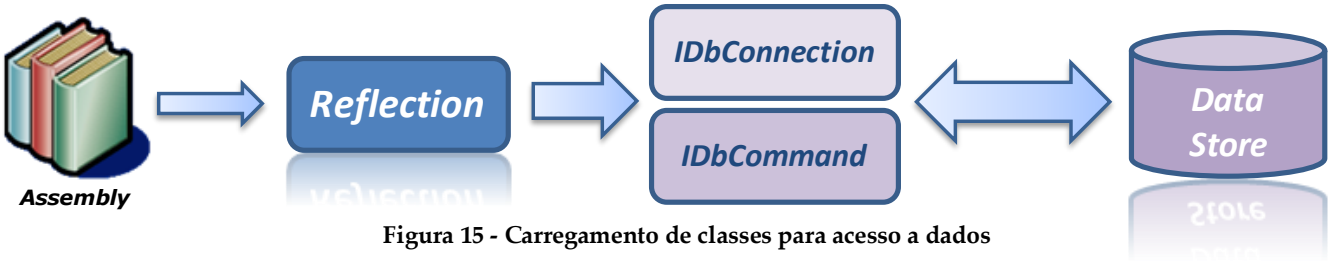


Figura 15 - Carregamento de classes para acesso a dados

O sistema não fornece nenhuma implementação destas interfaces, delegando essa implementação aos *providers*[14] de acesso a dados definidos em cada um dos indicadores, mas tem a responsabilidade de, em tempo de execução, encontrar e carregar a *assembly* do *provider* correcto e utilizar os tipos de dados que concretizam estas interfaces. Este mecanismo permite criar uma lógica de *plug-in* na camada de acesso a dados, tornando a aplicação mais flexível no sentido em que, sem efectuar alterações no código existente, possibilita a comunicação e extracção de informação, em abstracto, de qualquer sistema de gestão de bases de dados. O carregamento dinâmico de componentes de acesso a bases de dados é efectuado com recurso ao mecanismo de *reflection* disponibilizado pela *framework .NET*. Este mecanismo permite, em tempo de execução, carregar e analisar os metadados de uma *assembly*, obtendo assim informação acerca dos tipos, propriedades e métodos que esta define e partindo deste conhecimento instanciar os tipos correctos e invocar os métodos pretendidos.

[14] Tipicamente fornecidos pelos fabricantes das tecnologias de bases de dados

Web services

No respeitante à extracção de dados a partir de *web services*, foi implementado um mecanismo que permite que o sistema consuma dados a partir de qualquer *web service*, sem alterações ou desenvolvimentos adicionais, utilizando as classes da *framework .NET* para importação de serviços, geração de código C# e compilação de código. Com este mecanismo, basta fornecer a especificação do *web service* (documento *WSDL*) e o sistema gera dinamicamente uma classe que actua como *proxy* entre o sistema e o *web service*. Posteriormente, usando os mecanismos de *reflection*, já referidos, a classe *proxy* é instanciada e as suas operações podem ser invocadas. A lista, apresentada abaixo, e a Figura 16 sumariam o mecanismo supracitado:

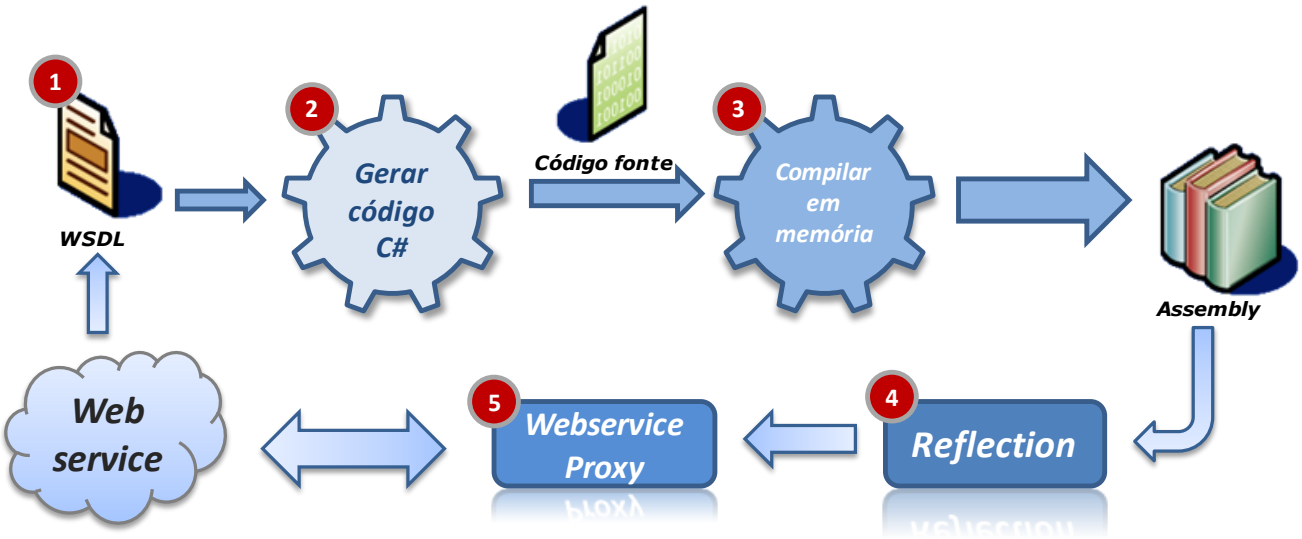


Figura 16 - Instanciação de *web service*

1. Ler o documento *WSDL*;
2. Gerar código C# que concretiza uma classe *proxy* para invocação das operações;
3. Compilar o código gerado em memória (obter uma *assembly*);
4. A partir da *assembly* e através do mecanismo de *reflection* instanciar a classe *proxy;*
5. Através do mecanismo de *reflection* invocar, dinamicamente, os métodos da classe *proxy* (operações do *web service*);

7.4.2 *Engine & Workers*

Os componentes *Engine* e *Worker* são os elementos basilares no processo de obtenção dos indicadores de desempenho. Esta subsecção visa detalhar as responsabilidades de cada um e a forma como cooperam para atingir este objectivo.

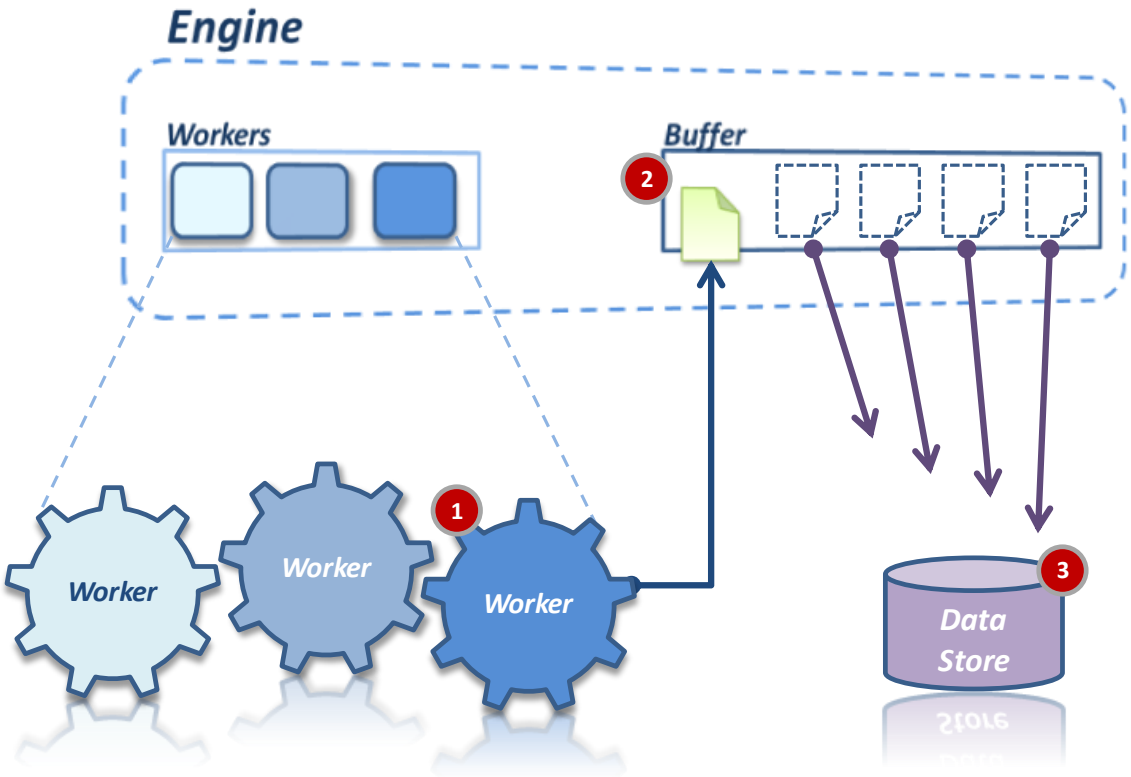


Figura 17 - Mecanismo de extracção de indicadores

O componente *Engine* tem a responsabilidade de, por um lado, no arranque do sistema consultar o repositório local no sentido de identificar e carregar todos os indicadores activos e respectivos filtros de pós-processamento de resultados, e por outro, de coordenar o mecanismo de obtenção de valores para estes indicadores. Para cada indicador activo é construída uma instância do componente *Worker*, que ficará responsável por obter os valores para esse indicador e aplicar os filtros de pós-processamento apropriados. O componente *Engine*, mantém uma lista com todas as instâncias *Workers* em execução e partilha com todos eles um *buffer*, onde estes últimos escrevem os resultados dos indicadores; possui ainda um mecanismo assíncrono que periodicamente

percorre, numa lógica de *FIFO*, o *buffer* de resultados e usando uma *thread pool*[15] persiste os valores em *cache* de forma paralela. Este mecanismo que se encontra esquematizado Figura 17 permite, por um lado, melhorar o desempenho e escalabilidade do sistema, uma vez que as escritas na *cache* são feitas periodicamente e em paralelo, e por outro, tornar o cálculo de valores para os indicadores independente da forma de construção da *cache* (*low-coupling* e *high-cohesion*).

O componente *Worker* funciona como um agente independente e autónomo, que incorpora uma definição de indicador e fica encarregue de periodicamente e segundo a definição do indicador em causa, recalcular os seus valores, aplicar os filtros de pós-processamento, caso existam, e escrever estes resultados no *buffer* partilhado com o componente *Manager.* A extracção de valores é feita através de um mecanismo de invocação assíncrona, que consiste em delegar a execução do comando de obtenção de dados ao respectivo repositório e, enquanto não chega a resposta, ficar em estado latente libertando assim o *CPU*. O *Worker* é "acordado" apenas quando chega a resposta e nessa altura retoma e conclui o processo. Este mecanismo contribui para um menor consumo de recursos e para o aumento da escalabilidade do sistema.

7.4.3 Pós-processamento de resultados

O pós-processamento de resultados é um dos mecanismos desenvolvidos para aumentar a flexibilidade, modificabilidade e extensibilidade do sistema e permite enriquecer os resultados dos indicadores através de transformações executadas por componentes especializados e auto-contidos (filtros). Estes componentes podem ser injectados, configurados e removidos, de forma dinâmica e sem impacto no restante funcionamento do sistema.

Em virtude da independência destes filtros do contexto em que executam é possível reutiliza-los noutros indicadores e/ou instalações do sistema. Com o intuito de incrementar a reutilização destes componentes, foi desenvolvida uma infra-estrutura que permite a sua configuração em tempo de execução. Para este efeito, foi reaproveitado o ficheiro *xml* usado pelas aplicações *.NET* para

[15] *Thread Pooling* é um mecanismo que permite optimizar a gestão de *threads* mantendo uma lista de *threads* que podem ser usadas pela aplicação e que voltam para a lista quando são descartadas. É especialmente útil para operações simples, em que o custo de criação e destruição da *thread* (que consome recursos do sistema) não compensa o trabalho efectivo que esta executa.

configurar alguns dos elementos de execução, criando secções específicas para a configuração dos filtros de pós-processamento de resultados.

Prova de conceito

Para validar a aplicabilidade deste mecanismo foram desenvolvidos, numa óptica de prova de conceito, dois filtros de pós-processamento:

- **AddURL**: este filtro adiciona, para cada linha de resultado, um *url* para outra aplicação, parametrizado com um identificador da linha de resultado. Desta forma, é possível aceder directamente a uma outra aplicação para visualizar informação complementar ao resultado apresentado. Por exemplo, considerando um indicador que retorna as últimas dez facturas por processar num departamento, este filtro pode adicionar um *url* para aplicação de visualização de facturas, permitindo visualizar directamente uma factura em concreto (passando o seu identificador).
- **FormatCurrency**: filtro que altera o valor de uma coluna, para o formato monetário. Esta formatação é feita consultando as definições de regionalização do sistema operativo em que está a ser executada a aplicação e aplicando o formato da região corrente, adaptando-se assim ao contexto de execução do sistema.

Configuração

Como foi previamente referido, os filtros de pós-processamento podem ser configurados através de secções específicas num ficheiro *xml*. A título de exemplo considere-se o filtro *addURL*, que aceita dois parâmetros de configuração: o nome da coluna que identifica a linha de resultado e o *url* base para a aplicação externa. Uma possível configuração está abaixo ilustrada:

```
<addURL>
    <properties>
      <add key="url" value="~/outraApp.aspx?ano="/>
      <add key="keyColumn" value="ano"/>
   </properties>
</addURL>
```

7.4.4 Indicadores Parametrizados

Os comandos de extracção de indicadores são definidos usando a pseudo-linguagem apresentada na subsecção 7.3.1 . Quando os indicadores são carregados é feita uma análise à expressão que define o comando do indicador, sendo substituído cada um dos filtros pelo seu valor por omissão, caso exista,

ou por uma tautologia[16] caso contrário, de forma a obter uma expressão válida e que não retorne resultados adulterados.

Quando é solicitado o valor de um indicador parametrizado: 1) O pedido é recebido pelo componente *Manager*, que solicita o resultado ao componente *Engine*; 2) Este, por sua vez, redirecciona-o para o componente *Worker* que concretiza a definição do indicador em causa; 3) Ao receber um pedido de indicador parametrizado, o componente *Worker* substitui os parâmetros recebidos na expressão original (fazendo de antemão validações de forma a prevenir ataques de *SQL Injection*) e submete o cálculo desse novo indicador ao respectivo repositório retornando os resultados obtidos.

7.4.5 Indicadores Relacionados

A funcionalidade de relacionar indicadores permite aos utilizadores finais do sistema, navegar na informação disponibilizada. O mecanismo desenvolvido é suficientemente genérico para permitir uma navegação na "horizontal", ou seja, complementar a informação de um indicador com outro que dê informação relacionada ou relevante, bem como navegação "vertical", isto é, ver informação mais geral ou mais detalhada de um determinado indicador, numa lógica de *drilling*. Como exemplo do conceito de navegação "horizontal", considere-se o seguinte caso: a partir de um indicador que dá informação sobre os clientes com maior volume de negócios, navegar para outro que dá detalhes sobre o historial de interacções com um destes clientes. Um possível exemplo de navegação "vertical", também conhecida como *drill*, prende-se com a possibilidade de, dado um indicador que fornece informação sobre totais anuais de vendas por artigo e por loja, visualizar informação das vendas efectuadas num mês em particular e partindo dessa informação, detalhar ainda mais e visualizar as vendas efectuadas num determinado dia.

As relações entre indicadores podem ser definidas pelos utilizadores finais e serão armazenadas no repositório local do sistema (ver **Modelo de dados**), sendo que uma definição de relação consiste em especificar um indicador de origem, um indicador de destino e um ou mais valores de entrada no indicador destino que permitam estabelecer a correlação. Do ponto de vista técnico, um pedido de visualização para um indicador relacionado traduz-se em calcular o valor do indicador destino com parâmetro(s) passado(s) como valor(es) de entrada (indicador parametrizado) obtidos do indicador de origem.

[16] Uma tautologia é uma expressão lógica que é sempre avaliada como verdadeira.

7.5 Armazenar Indicadores

O segundo passo no fluxo de informação do sistema ***TrueKPI*** consiste no armazenamento dos resultados obtidos na fase de extracção e subsequente construção de uma *cache* que mantém valores pré-calculados para todos os indicadores activos. A utilização do mecanismo de *cache* é fundamental para que a aplicação responda de forma célere a pedidos de consulta de indicadores. A *cache* de valores é concretizada na base de dados local de suporte, sendo o seu processo de construção da responsabilidade do componente *Engine*. Esta secção pretende mostrar quais os componentes envolvidos e quais as estratégias usadas para implementar esta funcionalidade.

7.5.1 Algoritmos de gestão da *cache*

Um indicador de desempenho pode ser visto, no limite, como o resultado de uma *query* que dá informação relevante a nível de negócio e nesse sentido o cálculo de indicadores consiste na invocação de operações em *webservices* ou na execução de *queries* nos sistemas origem e na recolha dos respectivos resultados. Desta definição de indicador resulta que o seu valor poderá, em rigor, ser representado através de uma tabela com um número variável de linhas e colunas. Um dos desafios apresentados na construção desta ferramenta foi desenhar um modelo de dados que suportasse o armazenamento de valores com diferentes estruturas. São então propostos dois algoritmos para o tratamento (leitura e escrita) dos valores dos indicadores na *cache* local:

Algoritmo baseado em serialização dos resultados:

Nesta abordagem, esquematizada na Figura 18, a tabela resultado do cálculo do indicador de desempenho é serializada e transformada numa estrutura *XML* que representa a tabela. Esta estrutura *XML* pode ser guardada num campo de texto da base de dados. Quando a informação deste indicador for solicitada, o sistema deve desserializar esta estrutura e reconstruir a tabela de resultados.

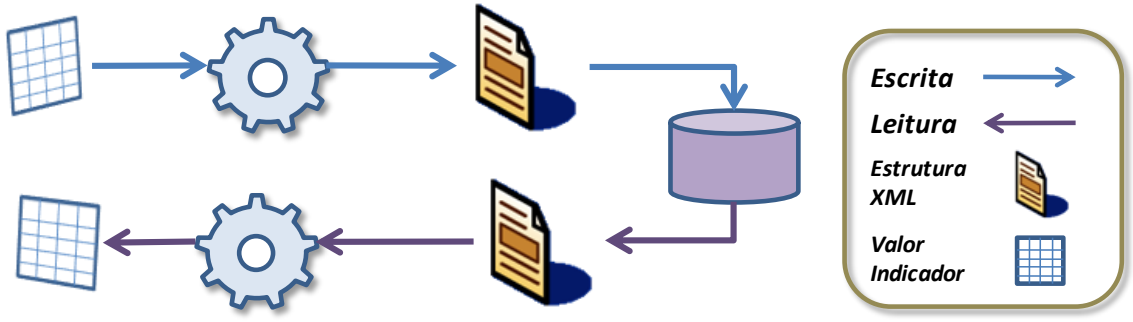


Figura 18 - Algoritmo baseado em serialização de resultados

Algoritmo baseado em modelo de dados dinâmico

Uma vez que, potencialmente, os valores de cada indicador têm uma estrutura diferente, esta solução propõe que cada indicador tenha uma tabela construída à sua medida e que respeite a sua estrutura de valores. Uma vez que não é possível saber, em fase de desenho/construção, que indicadores serão criados e consequentemente quais as estruturas que deverão ter as tabelas de valores, estas serão criadas dinamicamente, em tempo de execução, quando o primeiro valor for calculado (ver Figura 19). Para este efeito, o sistema gera, automaticamente, com base na estrutura dos resultados o comando *SQL*, para criar a tabela de *cache* e posteriormente as expressões adequadas para inserção dos resultados na respectiva tabela.

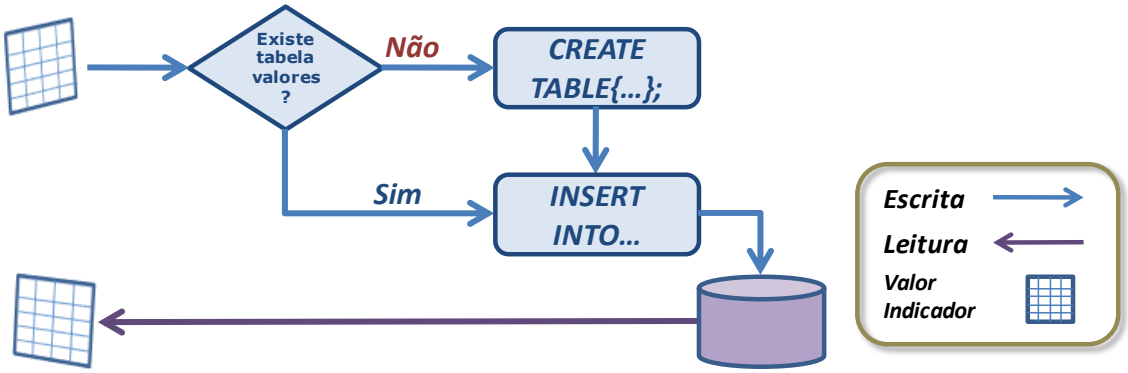


Figura 19 - Algoritmo baseado em modelo de dados dinâmico

Os algoritmos previamente apresentados, foram concretizados através das classes *XMLValuesManager* e *DynamicDataValuesManager* que suportam os algoritmos de **serialização de dados** e **modelo de dados dinâmico**, respectivamente, e implementam a interface *IValuesManager* (padrão de desenho *Strategy*).

7.5.2 Serialização de resultados *versus* modelo de dados dinâmico

Recordando que um alto desempenho é um dos atributos de qualidade pretendidos para o sistema desenvolvido, foi realizado um estudo comparativo entre os dois algoritmos de gestão de cache de forma a decidir qual o que melhor se adequa aos objectivos delineados. Este, consistiu em medir os tempos de escrita e de leitura de ambos algoritmos para diversos cenários de utilização (casos de utilização comum e casos extremos). As medições foram realizadas através da *framework* de testes *nUnit*, criando casos de teste para os diferentes

cenários, de forma a isolar e medir apenas a actuação dos componentes em estudo e recolhendo os tempos de execução (em segundos). Estas provas de desempenho permitiram ainda demonstrar os ganhos de desempenho obtidos pela utilização de uma *thread pool* que paraleliza as escritas que persistem os valores dos indicadores em cache (ver secção 7.4.2).

	Linear		***Threads***		***Variação linear/threads***		***Variação XML\Dynamic data (threads)***
Nº Escritas	***XML***	***Dynamic Data***	***XML***	***Dynamic Data***	***XML***	***Dynamic data***	
1	0,794142012	0,49971856	0,5207488	0,4556552	34,43%	8,82%	12,50%
10	0,860047392	0,92432912	0,48569838	0,46667104	43,53%	49,51%	3,92%
100	4,54353328	13,32215632	0,5107344	0,49771568	88,76%	96,26%	2,55%
1000	39,30251424	127,1618498	0,670492224	0,55880352	98,29%	99,56%	16,66%
Média	**11,375059**	**35,4770134**	**0,54691845**	**0,4947114**	**66,25%**	**63,54%**	**8,91%**

Tabela 5 – Resultados das medições para operações de escrita

A Tabela 5 mostra os resultados obtidos[17] para operações de escritas em cada um dos cenários propostos, bem como as variações entre os dois algoritmos e entre a aplicação do mecanismo de *threads* versus escritas sequenciais. Na Tabela 6 apresentam-se os tempos (em segundos) obtidos por cada um dos algoritmos para 100 operações de leitura[17].

Nº de leituras	***XML***	***Dynamic Data***	**Variação**
100	3,7994278	1,155603504	**69,58%**

Tabela 6 – Resultados das medições para operações de leitura

Ponderando sobre os resultados acima evidenciados, fica patente que:

- A utilização do mecanismo de *threads* permite melhorar o desempenho das escritas em cerca de 65%;
- Quando usado em conjunto com o mecanismo de *threads* o algoritmo de **modelo de dados dinâmico** é cerca de 9% mais eficiente que o algoritmo baseado em **serialização de resultados**, para operações de escrita;
- O algoritmo de **modelo de dados dinâmico** é cerca de 70% mais eficiente para operações de leitura.

[17] Cada resultado apresentado corresponde a um valor médio obtido a partir de dez ensaios. Encontra-se em anexo uma tabela com todos os valores obtidos.

O estudo que esta secção documenta permite-nos concluir que o algoritmo de **modelo de dados dinâmico** é, indubitavelmente, a opção que melhor se enquadra com os objectivos requeridos a nível de desempenho. Por outro lado, o modelo de dados que o suporta permite utilizar a *cache* de valores dos indicadores como uma fonte de dados para novos indicadores, criando assim uma lógica de **meta-indicadores**, ou seja, indicadores baseados em resultados de outros indicadores de negócio recolhidos. Este mecanismo vai permitir por exemplo, fazer análise histórica de resultados, perceber tendências e criar indicadores que relacionam dados oriundos de diferentes fontes.

7.6 Disponibilizar Indicadores

O fluxo de dados do sistema ***TrueKPI***, termina com a disponibilização da informação obtida nos passos anteriores para consumo da camada cliente e no cumprimento de um dos requisitos iniciais: possibilidade de visualizar e interagir com esta informação de formas distintas. Esta secção detalha de uma perspectiva técnica, os componentes, tecnologias e abordagens empregues para concretizar estes objectivos e implementar a camada cliente do sistema ***TrueKPI.***

7.6.1 *Web service*

A forma fundamental de disponibilização e interacção com os indicadores de negócio é através do *web service* que a camada lógica expõe; esta opção arquitectural melhorou a interoperabilidade do sistema, permitindo que sejam desenvolvidos clientes simples ou complexos em qualquer tecnologia, desde que tenha a capacidade de consumir dados através de *web services*, permitindo que o sistema continue a evoluir, embora mantendo a compatibilidade com camadas cliente actuais. A arquitectura desenvolvida deixa ainda aberta a possibilidade de evolução para outros modelos modernos de disponibilização de serviços como *cloud computing* ou a integração deste sistema como um serviço num macro-sistema concebido segundo padrões arquitecturais associados ao modelo *SOA.*

7.6.2 *Webpart*

Um dos elementos desenvolvidos para a camada cliente do sistema ***TrueKPI*** foi uma *ASP .NET WebPart* que suporta a visualização e interacção com um indicador de desempenho. A tecnologia de *WebParts* é a implementação da *Microsoft* na *framework ASP .NET* do conceito de *portlet* , que

consiste em componentes de visualização de informação independentes que podem ser adicionados, personalizados ou removidos pelos utilizadores finais de um portal ou de uma página *web* com o papel de hospedeira para estes componentes. Do ponto de vista técnico, os *portlets* funcionam em aplicações web que geram conteúdo HTML dinâmico disponibilizando métodos para geração de código HTML que representa o seu conteúdo e estado. Estes métodos são invocados aquando da geração da página de resposta a uma acção na aplicação hospedeira, sendo o HTML gerado agregado à resposta. Esta arquitectura permite obter um cliente final que cumpre com os requisitos apresentados a nível de personalização do *dashboard* de indicadores.

Durante o processo de desenho da arquitectura da camada cliente, foram identificados dois pontos de variabilidade, nomeadamente, a biblioteca usada para gerar as representações gráficas e a classe que serve de *proxy* para o *web service*. Nesse sentido foram empregues as estratégias, anteriormente discutidas, para acomodar estes factores e atingir os requisitos de qualidade propostos, obtendo-se assim a arquitectura esquematizada na Figura 20.

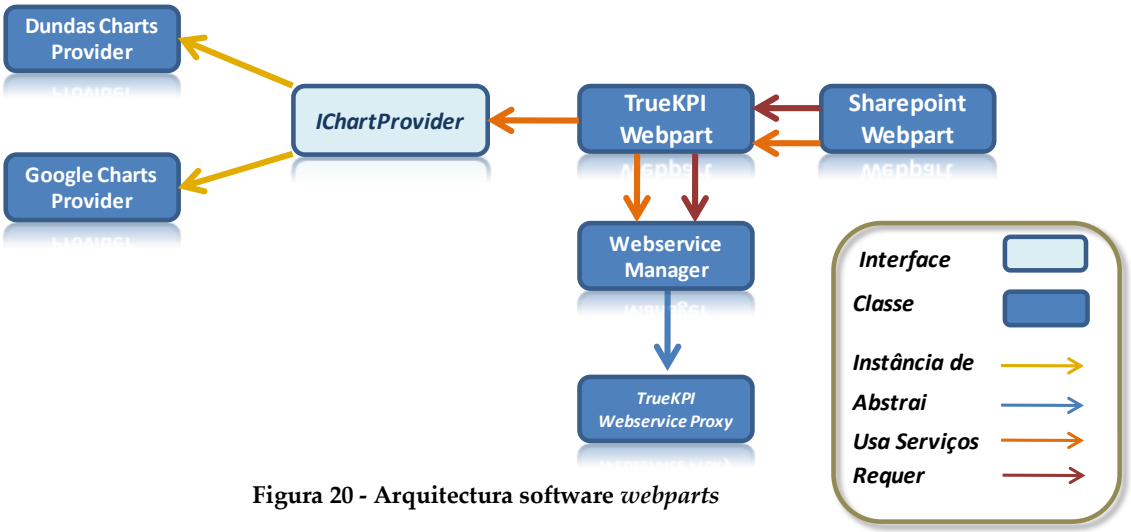


Figura 20 - Arquitectura software *webparts*

No decorrer do processo de construção foram testadas duas bibliotecas para gerar as representações gráficas dos indicadores de negócio. Inicialmente foi utilizado um componente que actua como *proxy* para o serviço de geração de gráficos do *Google* – *o Google Charts* –, (ver 11.6.2) tendo sido depois utilizada, como prova de conceito, uma outra tecnologia de geração de gráficos, o *Dundas Charts* que oferece mais funcionalidades e maior flexibilidade na construção dos gráficos.

7.6.3 *Dashboard*

O *dashboard* de visualização e interacção com os indicadores de desempenho fornecidos pela camada lógica do sistema, foi implementado criando uma aplicação *web* em *ASP .NET* que, consistindo numa página *ASP .NET WebParts Page*, serve de hospedeira a um conjunto de *webparts* que representam diferentes indicadores. Esta página pode ser personalizada e configurada pelos utilizadores finais, quer no que diz respeito aos indicadores a apresentar, quer ainda à sua disposição física e forma de representação gráfica, através dos métodos oferecidos para adicionar, remover e configurar *webparts*. A usabilidade foi enriquecida com técnicas de *AJAX*[18] para melhorar a experiência de utilização do *dashboard* (ver 11.6.2).

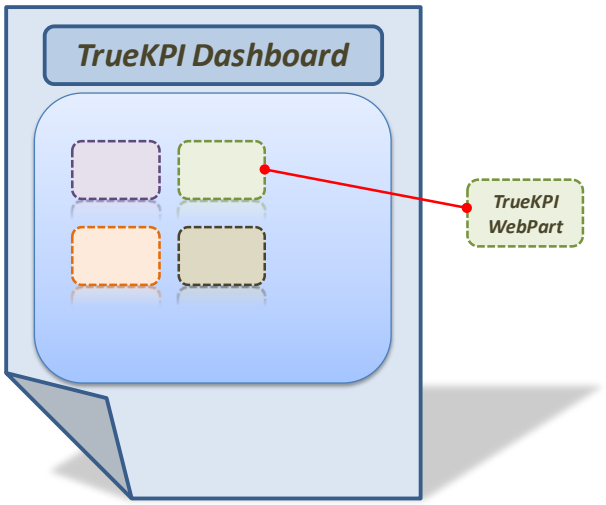


Figura 21 - Dashboard de indicadores

7.6.4 *Windows Sharepoint Services*

A infra-estrutura de criação de portais e aplicações *web Sharepoint*, suporta a tecnologia de *webparts*; no entanto as *webparts* desenvolvidas na tecnologia *ASP .NET* não são inteiramente compatíveis com as aplicações desenvolvidas em *Sharepoint*, com suporte para este tipo de controlo. Neste caso concreto, as *webparts* têm de ser construídas estendendo uma classe específica disponibilizada na *API* de desenvolvimento para *Sharepoint*. Este contratempo, foi resolvido criando um componente que estende a classe correcta para

[18] AJAX consiste num conjunto de técnicas de programação *web* que permite criar aplicações ricas e interactivas, através da execução de pedidos assíncronos ao servidor e actualizações parciais da página de interface. Estas técnicas permitem melhorar o desempenho da aplicação e a experiência de utilização fazendo pedidos mais simples e actualizando apenas as partes relevantes da página *web*.

webparts em *Sharepoint* e que contém uma referência para o componente *TrueKPI WebPart*, ao qual é delegada a tarefa de gerar o código HTML que mostra a informação do indicador de desempenho (ver Figura 20).

7.6.5 *Windows Mobile*

A criação de um cliente rico para a plataforma *Windows Mobile*, não estava prevista nos objectivos iniciais do projecto, no entanto tendo em conta que no actual contexto social e tecnológico a computação ubíqua e a mobilidade têm uma relevância não menosprezável, foi desenvolvido um protótipo muito simples, cuja arquitectura se encontra esquematizada na Figura 22. Este componente demonstra ainda a interoperabilidade do sistema ***TrueKPI*** e as vantagens de ter um sistema descentralizado e estruturado numa lógica de *low-coupling*, na criação de novos componentes e na evolução do sistema. Este cliente permite consumir os indicadores de desempenho através do *web service* disponibilizado pelo sistema e possui um mecanismo de *cache* que armazena os seus valores numa base de dados local do terminal (ver 11.6.2), de forma a possibilitar a consulta de indicadores mesmo em ambiente desligado (independente da existência de conectividade). No que diz respeito à geração das representações gráficas, foi utilizada com sucesso a biblioteca de geração de gráficos *Resco Compact Chart*.

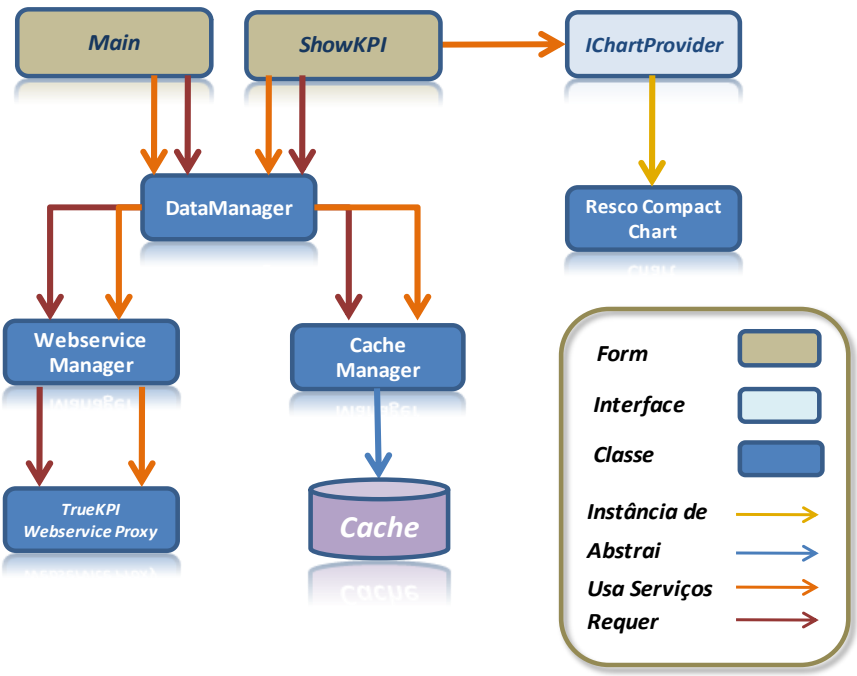


Figura 22 - Arquitectura cliente *Windows Mobile*

7.7 Aspectos

O desenvolvimento orientado a aspectos (*AOP - Aspect Oriented Programming)* é um paradigma de programação e desenvolvimento de *software*, emergente, que visa solucionar alguns problemas de modularização causados pela abordagem proposta pelos paradigmas tradicionais (procedimental, orientado a objectos, etc.) para decomposição de um problema em unidades de implementação (módulos, classes e métodos). Neste tipo de aproximação, um problema é dividido em unidades lógicas e funcionais que contribuem para a sua resolução. No entanto, qualquer sistema não trivial tem um conjunto de funcionalidades que são transversais a várias unidades de implementação, por exemplo: medir o progresso de uma operação constituída por *n* passos, validar coerência de dados ou fazer registo de situações anómalas. Para esta classe de funcionalidades, a abordagem tradicional tem uma resposta parca e conduz a uma perda de modularidade causada pela necessidade de "espalhar" código repetido em diferentes unidades de implementação (registar excepções em todas as operações em que estas podem ocorrer) e de misturar responsabilidades diferentes na mesma unidade (o componente que oferece uma operação ter a responsabilidade de validar a identidade do invocador). A abordagem proposta pela *AOP* consiste em encapsular as funcionalidades transversais em unidades de implementação, denominadas **aspectos** e definir **pontos de injecção** (***pointcuts***), ou seja regras que especificam onde serão injectados os aspectos (antes, durante ou depois da invocação a um método, em métodos com determinadas características, etc.). Os aspectos serão executados nos pontos de injecção definidos, criando assim um **ponto de ligação** (***join-point***), entre o aspecto e o método original, e concretizando a funcionalidade pretendida.

Na secção seguinte será apresentada a *framework* que suportou a implementação de funcionalidades sob a forma de aspectos – *Spring .NET Framework*. Posteriormente serão descritos os aspectos desenvolvidos e as funcionalidades que implementam.

7.7.1 *Spring .NET Framework*

A *Spring .NET Framework* é a implementação para *.NET* da *Java Spring Framework* e consiste numa plataforma constituída por diferentes módulos, que disponibiliza soluções que resolvem ou ajudam a resolver problemas recorrentes em aplicações de âmbito empresarial, como por exemplo, acesso a dados, gestão de transacções, filas de mensagens, entre outros. No contexto

deste trabalho foram utilizadas as infra-estruturas de *Inversion of Control* (*Dependency Injection*) e *AOP*, módulos *Spring.Core* e *Spring.AOP*, respectivamente, permitindo obter uma solução que utiliza aspectos e que, por intermédio de *Dependency Injection*, pode ser configurada em tempo de execução. Através do ficheiro de configuração da aplicação são parametrizadas as classes concretas que serão utilizadas para implementar determinadas interfaces (injecção de dependências), bem como são configurados e geridos os aspectos e pontos de injecção, promovendo a modificabilidade e extensibilidade do sistema.

7.7.2 Registo de anomalias (excepções)

Apesar do sistema *TrueKPI* ter sido desenhado e construído de forma a atingir elevados padrões de qualidade, não é viável testar todas as possibilidades de execução, sendo portanto natural que ocorram situações inesperadas que levem a comportamentos anómalos. Nestes casos, é desejável ter um mecanismo que registe as anomalias, no sentido de facilitar a sua depuração e correcção. Para esse efeito, foi desenvolvido um aspecto que faz o registo, numa tabela da base de dados local do sistema, de excepções não tratadas que escalem até ao componente *Manager*. *A posteriori*, foi criado um indicador, ilustrado na Figura 33, que permite monitorizar essa tabela e fornecer dados sobre eventuais situações de erro que tenham ocorrido no sistema.

7.7.3 Recomendações para novos indicadores

O sistema *TrueKPI* tem por base um mecanismo de cache que armazena valores pré-calculados para os indicadores de negócio definidos. No entanto, este só é aplicado aos indicadores originais, ou seja, sempre que é solicitado um indicador parametrizado, esse valor terá de ser calculado na fonte de dados original acarretando eventuais problemas de desempenho.

Para mitigar esta limitação do sistema, foi desenvolvido um aspecto que regista numa tabela as invocações ao método que calcula indicadores parametrizados, guardando o identificador do indicador invocado, os parâmetros passados e a data de invocação, tendo sido definido para este aspecto, um ponto de injecção antes da invocação do método. Com os dados recolhidos é possível obter informação estatística de quais são os indicadores parametrizados mais invocados (ver Figura 33) e, caso se justifique, criar um

novo indicador com os parâmetros pretendidos (usufruindo assim do mecanismo de *cache*).

7.7.4 Autorização

O cálculo de indicadores de negócio permite expor dados relevantes a nível de gestão que, pela sua natureza, podem ser de acesso restrito. De igual forma diferentes utilizadores poderão ter diferentes níves de acesso no que diz respeito à configuração do sistema. Nesse sentido é impreterível a existência de um mecanismo de autorização que regule quem tem acesso a que operações e a que conjuntos de dados. Com base nestas premissas, foi desenvolvido um mecanismo que consiste numa interface genérica para *providers* de autenticação e um aspecto que utiliza esta interface para autorizar a execução de determinadas operações.

Interface de autorização

A autorização de acesso a recursos é efectuada por intermédio de uma interface que apenas define um método que recebe o nome da operação, os parâmetros da invocação e um *token* de autenticação genérico (*array* de *bytes*), retornando um valor booleano que indica se a operação é ou não autorizada. O *token* de autenticação permite que cada classe que implementa a interface, decida que informação precisa para fazer autorização: o nome de utilizador (assumindo que este já está autenticado), nome de utilizador e palavra-passe, um *cookie* ou qualquer outro. A infra-estrutura de *Dependency Injection* permite definir e alterar, em tempo de execução, sem fazer alterações ou recompilar o sistema, qual a classe que será usada para concretizar a interface de autorização.

Aspecto

Em complemento à interface mencionada, foi desenvolvido um aspecto que encapsula a funcionalidade de autorização utilizando os serviços desta para autorizar ou impedir o acesso a determinadas operações, dando origem a uma excepção neste último caso. O emprego de *AOP* para implementar o mecanismo de autorização vai permitir, obter uma grande flexibilidade e granularidade na definição de regras de acesso ao sistema, no sentido em que é possível definir pontos de injecção (onde será executado o código do aspecto) através de expressões regulares, atributos dos métodos e valores dos seus parâmetros. Deste modo é possível definir, por exemplo, operações que não estão sujeitas a autorização ou outras que apenas necessitam de autorização em certas condições.

A utilização destes dois componentes permitiu concretizar um mecanismo de autorização extremamente poderoso, no sentido em que é:

- Adaptável ao ponto de suportar diferentes mecanismos e tecnologias de autenticação e de poder facilmente permutar entre estes;
- Granular, oferecendo um grande controlo sobre que operações e em que condições estão sujeitas a autenticação, através da definição de pontos de injecção;
- Flexível, permitindo, caso se justifique, configurações mais avançadas que usam múltiplas implementações da interface de autorização para diferentes operações.

7.8 Qualidade de Software

O aumento da oferta e crescimento do mercado de soluções informáticas, levou a um aumento das expectativas em relação a estas soluções, não só ao nível das funcionalidades disponibilizadas, mas principalmente ao nível da qualidade do software. Na Truewind têm sido dado um enfoque cada vez maior na disponibilização de soluções bem testadas e cumprindo padrões de qualidade exigentes. Permitindo, por um lado, aumentar a satisfação e confiança dos seus clientes, propiciando a angariação de novos projectos ou contratos, e por outro, minimizando em projectos que seguem o modelo de *managed services*[19] o risco, inerente à falha das aplicações e sistemas, assumido por conta do cliente. Nestes projectos, em caso de falha, é responsabilidade da Truewind a execução de intervenções no sentido de solucionar os problemas de forma célere, que podem implicar muitas vezes a alocação de recursos escassos, que nestes períodos não estão a gerar valor para a empresa. Estes factores, bem como a necessidade de contribuir com novas práticas e ferramentas para a execução de diversos projectos em curso na empresa, motivaram a que aplicação fosse desenhada e construída de forma a atingir elevados padrões de qualidade, tendo estes sido obtidos através da utilização de um processo de desenvolvimento bem estruturado e sustentado em testes automatizados que verificam e validam cada versão da aplicação. No sentido de garantir uma efectiva qualidade dos testes efectuados foram considerados diversos factores, quer no desenho da aplicação, quer na sua construção que se enumeram ao longo dos próximos parágrafos.

[19] Modelo de contratação em que a empresa que oferece um serviço fica responsável por gerir e administrar não só o serviço, como toda a infra-estrutura que o suporta com base num nível de serviço pré-estabelecido.

Um teste unitário é tanto mais efectivo quanto maior a percentagem de caminhos de execução que exercita. Nesse sentido quanto maior o número de possibilidades de execução de uma unidade de código, mais difícil é garantir a efectividade dos testes que a verificam. Com base nesta premissa, o código foi estruturado em métodos simples, contidos e com objectivos precisos, com o menor número de parâmetros possível e utilizando estruturas de dados *fortemente tipadas*. Foram utilizadas, sempre que plausível, enumerações e tipos de dados com a menor quantidade de valores mas ainda suficientemente expressivos para implementar a semântica pretendida (por exemplo, usando uma representação de 16 *bits* para armazenar inteiros, cujos valores não excedam -32768 no limite inferior e 32767 no limite superior).

Foram definidos construtores adicionais para algumas classes, utilizando o padrão de desenho *dependency injection*, com o propósito de instrumentar os objectos em execução de forma poder avaliar o estado interno destes, antes e depois da execução dos testes e verificar da sua correcção.

As baterias de testes desenhadas são compostas por testes de "caixa negra", obtidos a partir dos requisitos funcionais e de qualidade e testes de "caixa branca", para exercitar as estruturas de código mais complexas e são automaticamente validadas durante o processo de construção através da ferramenta *nCover*. Esta ferramenta gera um conjunto de relatórios que permitem avaliar a complexidade[20] de cada método e as baterias de testes segundo os seguintes critérios de cobertura[21]:

- **Cobertura de símbolos**: Quantidade de símbolos de depuração percorridos;
- **Cobertura de métodos**: Quantidade de métodos invocados;
- **Cobertura de ramos de execução**[22]: Quantidade de estruturas de verdadeiro/falso avaliadas para ambos os casos.

Para testar a interface de utilizador do *dashboard* de indicadores, foram criadas aplicações, utilizando a *framework Watin*, que automatizam conjuntos de operações sobre a interface. Apesar da execução destas aplicações de teste não estar automatizada e integrada no processo de construção (servidor de integração), serve como complemento aos testes unitários e permite verificar a correcção da interface com utilizador.

---

[20] Avaliando o número de caminhos independentes de execução de um método. Esta informação auxilia a inferir sobre o número mínimo de testes necessários para cumprir o critério de cobertura de ramos de execução [22].

[21] Ver [17] para uma discussão mais detalhada sobre de critérios de cobertura de testes.

[22] *Branch Coverage.*

Capítulo 8

Resultados

Neste capítulo são apresentados os resultados alcançados no âmbito do Projecto em Engenharia Informática, quer no que concerne ao processo de desenvolvimento identificado e adoptado, quer quanto ao sistema desenvolvido. Discutem-se ainda os modelos de desenvolvimento propostos para a criação de soluções de *business intelligence* recorrendo ao sistema ***TrueKPI***, o modelo de negócio aplicável, bem como a sua adequabilidade ao paradigma da organização de acolhimento.

8.1 Processo de desenvolvimento

8.1.1 Integração Contínua

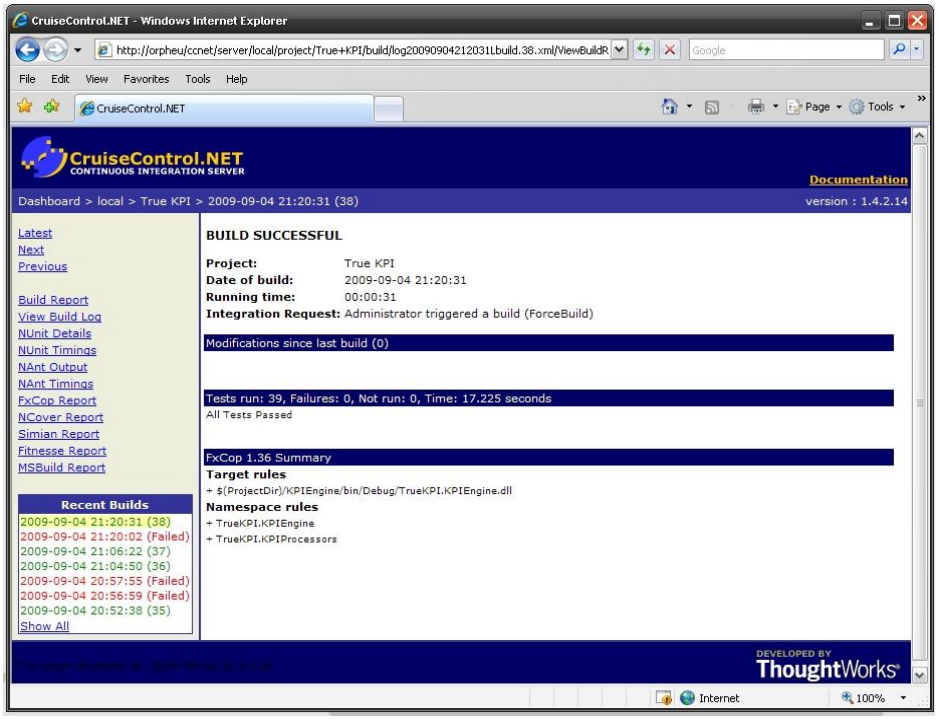


Figura 23 - Painel de controlo servidor de integração

O sistema ***TrueKPI*** foi inteiramente desenvolvido através da metodologia apresentada no Capítulo 3. Para suportá-la foi utilizado o servidor de integração *CruiseControl .NET* que automatiza o processo de construção, verificação e validação de uma versão do produto, para cada integração de novas funcionalidades (ou alterações na base de código em geral). Este processo de integração de alterações culmina com a geração de um conjunto de relatórios que dão informação sobre os erros (em caso de falha), cobertura atingida pelas baterias de testes e recomendações resultantes da análise estática de código.

8.1.2 Qualidade de Software

O controlo da qualidade do produto desenvolvido foi assegurado através da execução automática de testes unitários e de integração que são aplicados no final de cada integração. Os relatórios produzidos permitiram controlar a complexidade média das classes desenvolvidas, mantendo-a a níveis baixos (1,38 de média e 4 de valor máximo) e assegurar a efectividade dos testes atingindo coberturas de:

- 61,77% para cobertura de símbolos
- 46,58% para cobertura de métodos
- 41,58% para cobertura de ramos de execução

Os relatórios de construção, ilustrados na Figura 24, permitiram ainda controlar a qualidade de código fonte, considerando e eliminando os problemas identificados pela ferramenta de análise estática de código *FxCop*.

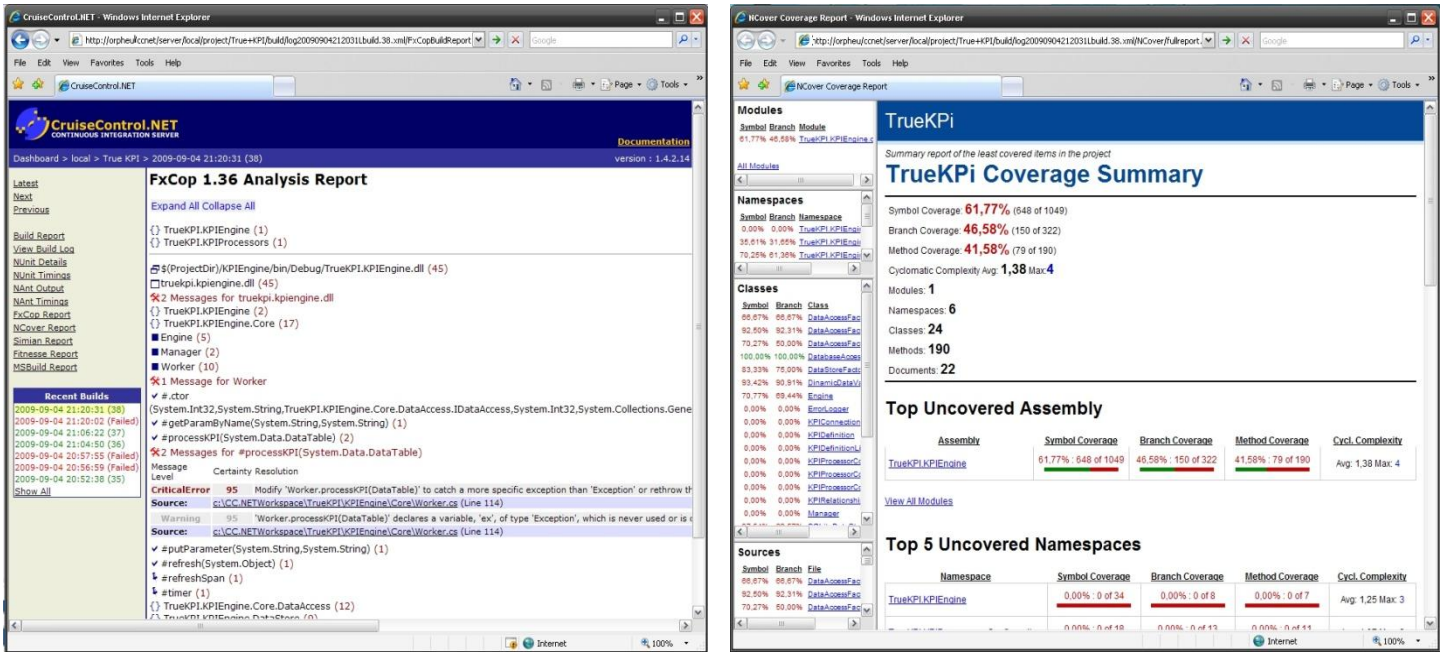


Figura 24 - Ferramentas de garantia da qualidade de software

8.1.3 Componentes reutilizáveis

Qualquer organização com alguma maturidade no processo de desenvolvimento de software tem a preocupação de desenvolver componentes que possam ser reutilizados noutras instalações e versões do produto ou noutros projectos e produtos, como forma de optimizar custos de desenvolvimento e agilizar o *time to market* de futuras soluções. O trabalho realizado permitiu a criação quer de componentes que poderão ser reutilizados em diferentes instâncias do sistema ***TrueKPI***, sendo disso exemplo os filtros de pós-processamento de resultados (ver 7.4.3), quer ainda de componentes que poderão ser utilizados noutros produtos ou soluções, como sejam os relacionados com o carregamento e instanciação dinâmica de classes, ou os de geração automática de classes *proxy* para interacção com *web services.*

8.2 Produto

Um dos objectivos propostos para o sistema ***TrueKPI*** consistia na possibilidade de disponibilizar soluções para clientes e contextos de execução diversificados, podendo, no limite, ser visto como uma linha de produto com os seguintes pontos de variabilidade: **fontes de dados**, **filtros de pós-processamento**, ***front-end*** e **mecanismo(s) de autorização**. Nesse sentido, para cada instalação do sistema deverão ser escolhidas e parametrizadas as peças adequadas para construir a solução pretendida.

8.2.1 Família *TrueKPI*

O sistema desenvolvido consiste numa solução descentralizada, composta por um servidor responsável por recolher os indicadores de negócio, disponibilizando-os através de um *web service*, e um conjunto de aplicações periféricas (ver Figura 27) que interagem com este na apresentação e exploração da informação disponibilizada, concretamente:

- Dois clientes *web:*
  - *Dashboard* materializado numa aplicação *ASP .NET;*
  - *Dashboard* consubstanciado num portal *Sharepoint*.
- Cliente simplificado em *Windows Mobile*.

8.2.2 Modelo de desenvolvimento

Os fluxogramas abaixo esquematizados ilustram o modelo de desenvolvimento proposto e utilizado na prova de conceito, para soluções ***TrueKPI***.

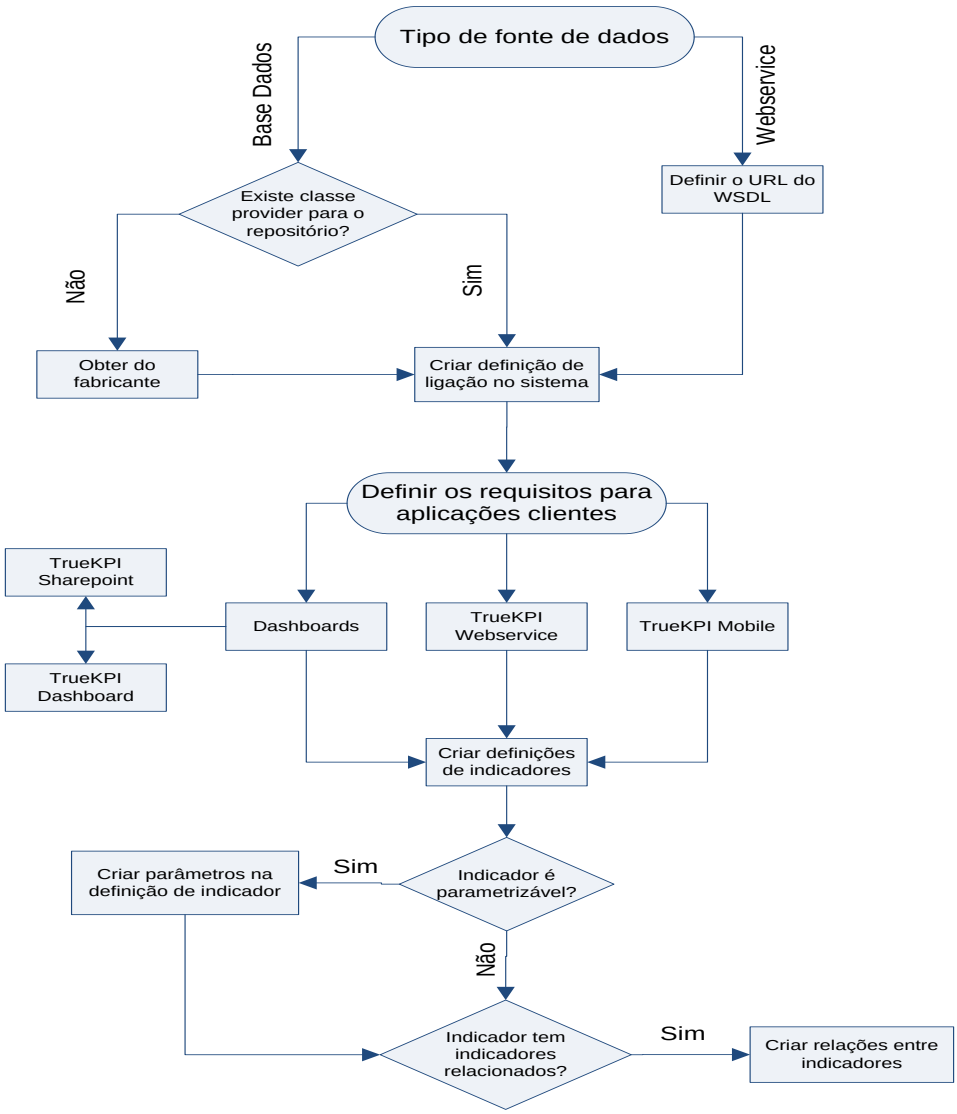


Figura 25 - Modelo de desenvolvimento (funcionalidade básica)

Como pode ver-se no fluxograma, ilustrado na figura acima, é possível desenvolver soluções de *business intelligence* à medida, através da ferramenta ***TrueKPI***, com esforço mínimo de configuração e sem necessidade de desenvolvimentos adicionais, sendo possível utilizar o sistema em ambiente *web* ou em contextos de mobilidade (***TrueKPI Mobile***). As instalações disponibilizadas segundo este modelo podem ser concretizadas por qualquer pessoa que conheça a linguagem *SQL*, mesmo não possuindo conhecimentos técnicos do sistema ou capacidades de programação. Este modelo, sendo bastante simples, permitirá oferecer as principais funcionalidades suportadas e será adequado para grande parte dos cenários de utilização previstos.

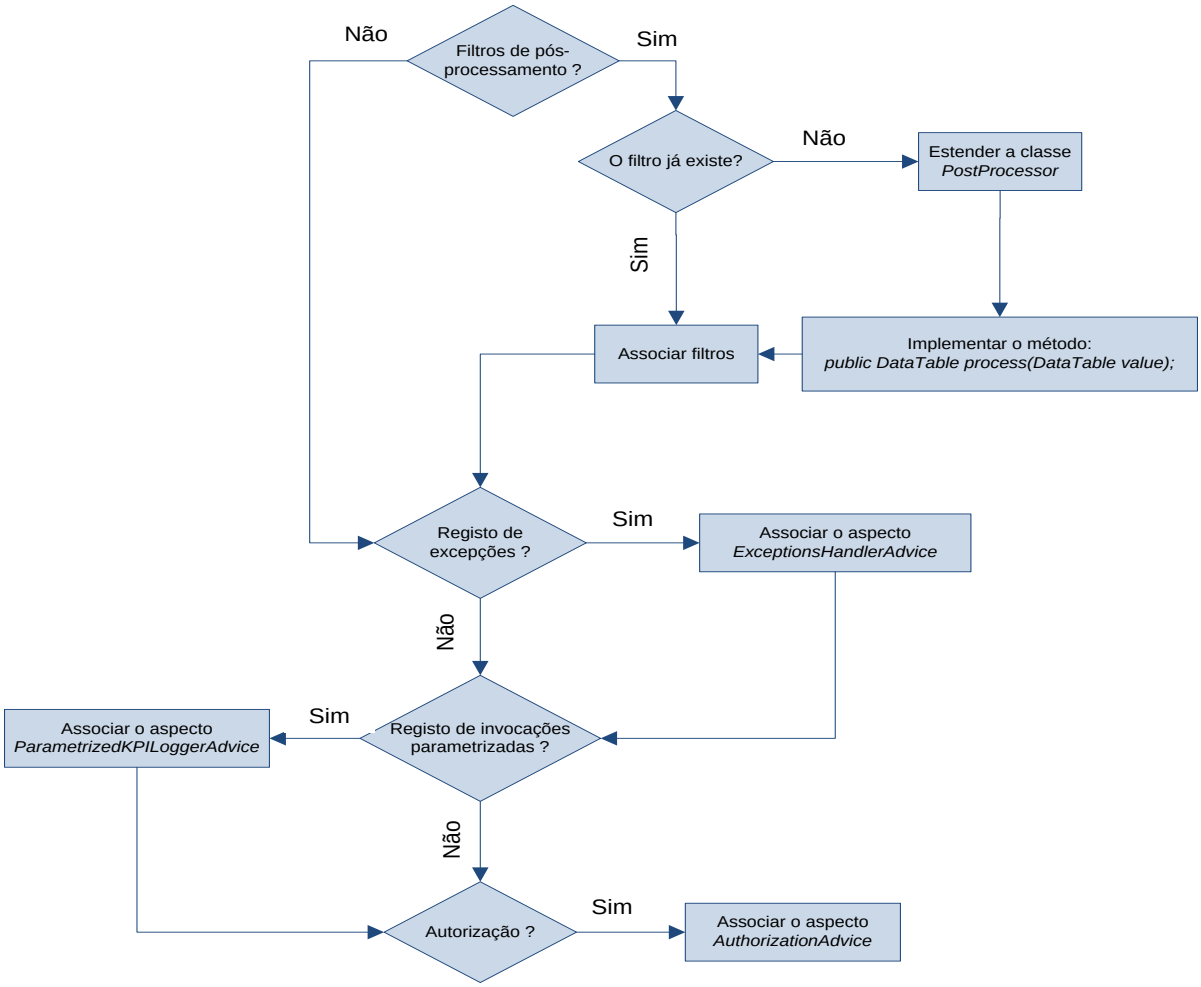


Figura 26 - Modelo de desenvolvimento (funcionalidade avançada)

Em casos mais específicos é possível com um baixo esforço de desenvolvimento, ou tendo a conta a reusabilidade de componentes, criar soluções mais complexas que respondem a requisitos particulares. O fluxograma, acima exposto, demonstra (partindo do modelo anterior) a evolução do sistema com o intuito de suportar filtros de pós-processamento (reutilizando algum componente previamente desenvolvido ou estendendo a classe *PostProcessor*) e utilizar funcionalidades adicionais concretizadas através de um paradigma orientado a aspectos. Neste último caso, existem já três aspectos responsáveis por gerir autorização de acesso ao sistema, registar excepções e invocações parametrizadas. No entanto, fazendo uso da infra-estrutura de gestão de aspectos discutida na secção 7.7 será possível injectar novas funcionalidades. Convém ainda referir que é possível desenvolver clientes de acesso ao sistema, à medida, tirando proveito do acesso via *web services* à camada de extracção de indicadores.

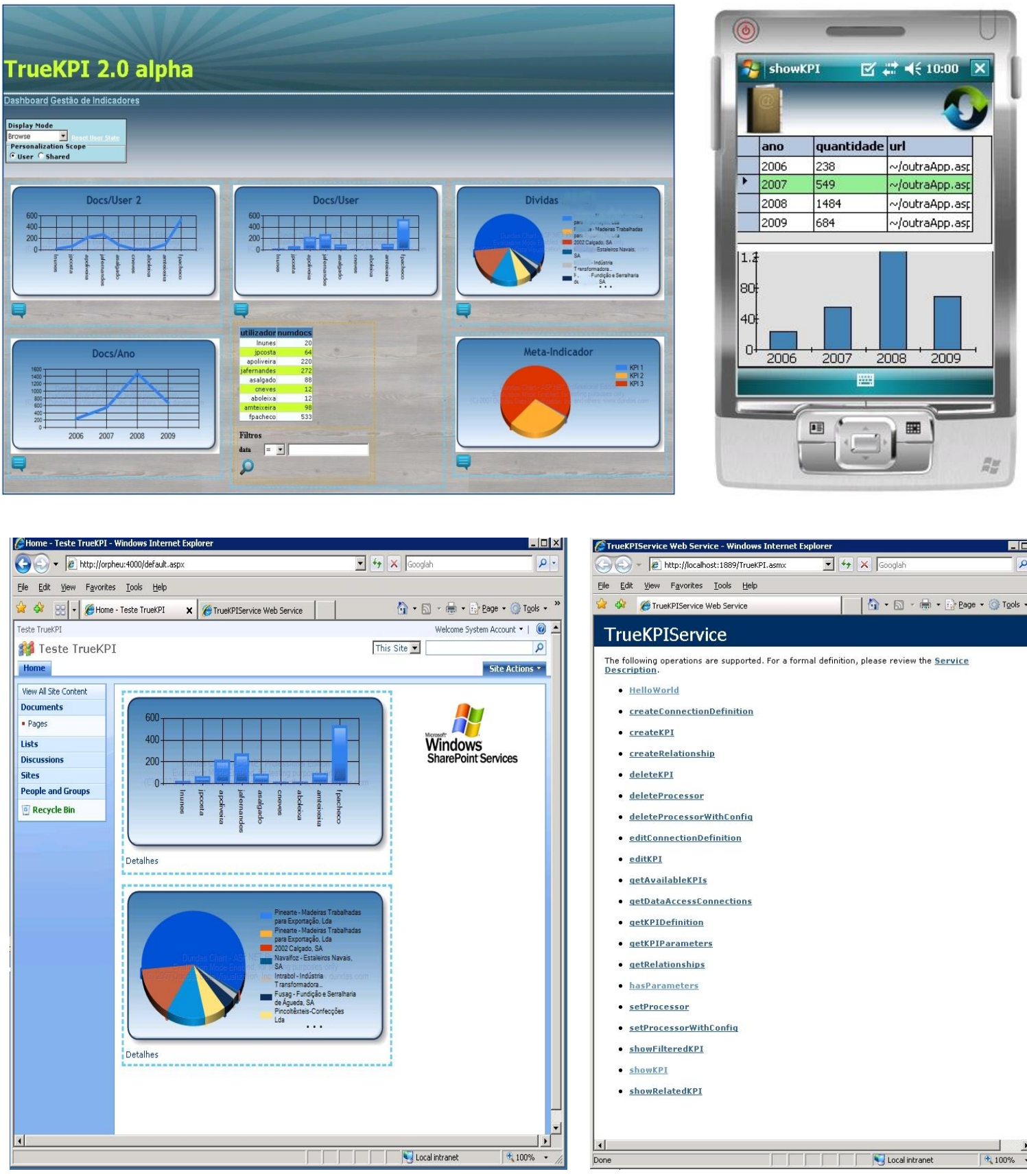


Figura 27 - Família *TrueKPI*

8.2.3 Modelo de negócio

Apesar do principal foco de actuação da Truewind estar centrado na disponibilização de soluções tecnológicas à medida, começa a ser dado um relevo cada vez mais significativo à prestação de serviços de consultoria funcional, baseada muitas vezes em produtos. É possível, desta forma, capitalizar os conhecimentos e experiência adquiridos em ferramentas de suporte ao negócio (*ERP, CRM,* entre outros) que actuam em áreas transversais a vários domínios, como por exemplo, a gestão financeira, a gestão de fornecedores, ou gestão de clientes. A ferramenta desenvolvida permite consolidar esta área providenciando uma nova ferramenta de gestão de baixo custo, rápida instalação e configuração a clientes que não disponham de aplicações com capacidades de *business intelligence*; permite ainda acompanhar a evolução das necessidades de cada cliente, adaptando a solução, em cada momento, aos seus requisitos e constrangimentos.

Sabendo que existem muitas organizações (inclusivamente de diferentes domínios) que suportam o seu negócio nas mesmas tecnologias de *ERP* será possível, em abstracto, reutilizar definições de indicadores para diferentes clientes, optimizando a prestação de serviços, mas também criar soluções para clientes que actuem em domínios em que a Truewind não tem experiência.

Apesar da aplicação desenvolvida não se propor fazer concorrência a outras soluções de *business intelligence* disponíveis no mercado, o facto de seguir os paradigmas do desenvolvimento ágil, apresentando um baixo custo de instalação e manutenção, suportando um rápido ciclo de apresentação de resultados, constitui uma grande mais-valia face às ferramentas tradicionais, que se consubstancia numa pronta resposta aos desafios muitas vezes apresentados pelos clientes da Truewind.

8.3 Objectivos atingidos

Observando os resultados obtidos conclui-se que foram atingidos com sucesso os principais objectivos ao nível do processo de desenvolvimento, no sentido em que foi identificado, implementado e utilizado um processo de desenvolvimento aplicável às práticas em uso na empresa permitindo, através de testes automáticos e análise estática de código, melhorar a qualidade da solução e optimizar a fase de construção, identificando atempadamente erros e problemas que esta possa apresentar.

No que diz respeito à solução desenvolvida, foram atingidos os objectivos definidos:

- A nível das funcionalidades da aplicação, validadas empiricamente e verificadas através de testes automáticos;
- A nível do cumprimento dos atributos de qualidade, validados de forma empírica e verificados por meio de testes automáticos (graças ao carácter operacional da definição do atributo e da especificação de medidas de resposta[23]).

Teria ainda assim sido desejável ter atingido uma maior cobertura de código nos diferentes critérios, mas limitações a nível de calendário impediram um maior investimento na escrita dos testes unitários e de integração exigíveis.

[23] Ver secção 4.2

Capítulo 9

Avaliação e Estudo Comparativo

O sistema ***TrueKPI*** tem por propósito agilizar o processo de desenvolvimento, instalação e manutenção de sistemas de geração e visualização de indicadores de negócio, permitindo a criação de *dashboards* definidos pelos utilizadores finais. A arquitectura proposta além de simplificar e acelerar a entrega de funcionalidades ao cliente final, propõe-se a atingir estes objectivos por uma fracção dos custos das aplicações tradicionais de *business inteligence*.

Neste capítulo apresenta-se um estudo comparativo simplificado, com base num caso prático descrito na secção 9.1.2 , incluindo a análise e comparação do processo e dos custos de desenvolvimento de um *dashboard* em ambiente *web*, que disponibiliza indicadores de desempenho, construído seguindo duas abordagens diferentes:

- **Tradicional**, utilizando plataformas disponibilizadas por grandes fabricantes, como a Microsoft ou a Oracle;
- **Ágil**, utilizando a ferramenta desenvolvida e as técnicas estudadas.

No final deste capítulo é igualmente apresentada uma análise comparativa entre o ***TrueKPI*** e o *SicGest*, aplicação que serviu de base a este processo de reengenharia.

9.1 Comparação com o modelo tradicional

9.1.1 Ciclo de desenvolvimento

Tipicamente, o desenvolvimento de uma infra-estrutura de suporte a aplicações de *business intelligence* segue uma abordagem *top-down* que inclui três grandes fases:

1. Levantamento dos indicadores e *queries* a serem suportados;
2. Desenho e construção do modelo de dados analítico;
3. Definição e implementação dos processos ETL que fazem a extracção e integração de informação a partir das fontes de dados de negócio.

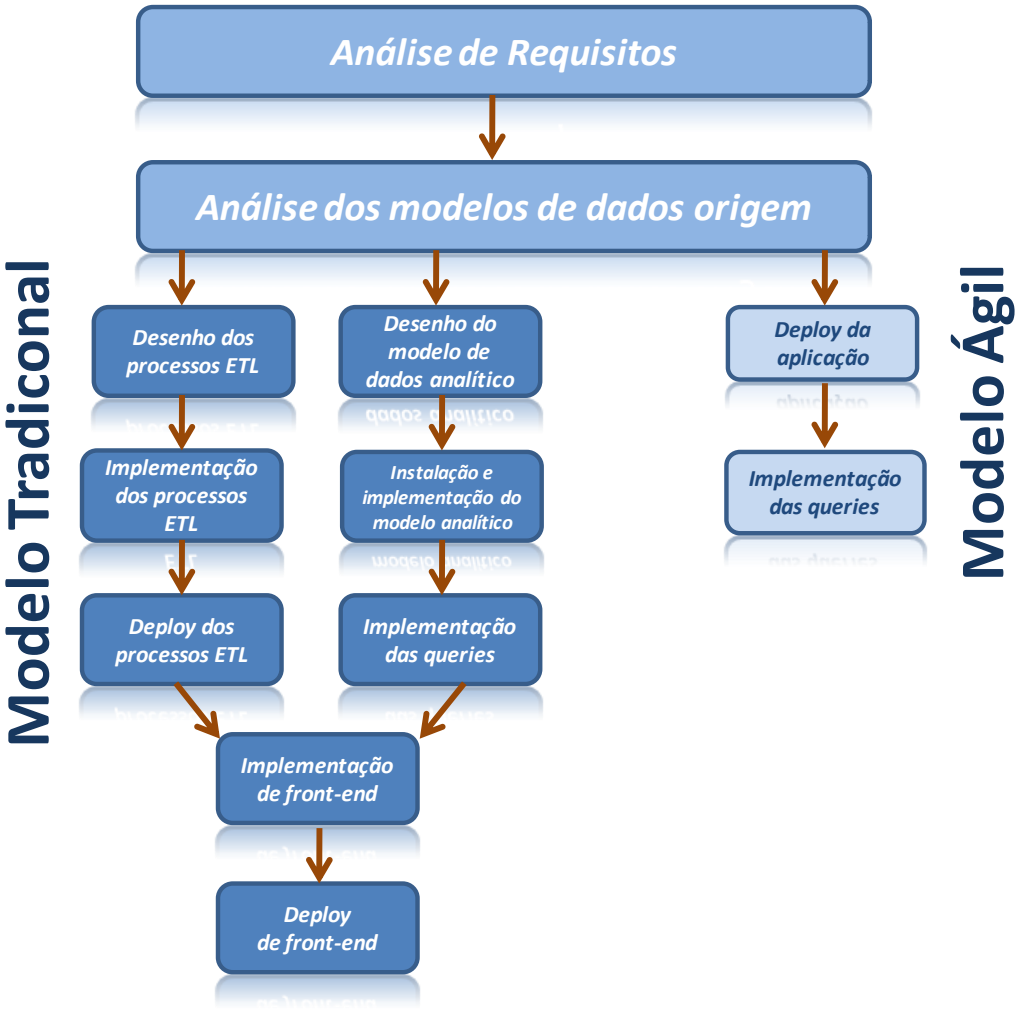


Figura 28 - Desenvolvimento de aplicações *Business Intelligence*

Só após a criação desta infra-estrutura é possível desenvolver aplicações que assentam no modelo analítico para consumo, apresentação e interacção com a informação. Na Figura 28 é apresentado o ciclo de desenvolvimento de uma aplicação desenvolvida seguindo este paradigma.

O ciclo de desenvolvimento proposto para aplicações criadas através da ferramenta ***TrueKPI*** (Figura 28) é bastante mais simplificado, sendo constituído apenas pelo levantamento dos indicadores e *queries* de suporte aos indicadores de negócio, que, tirando partido da capacidade das fontes de dados de origem, serão executadas directamente nos respectivos repositórios, evitando a implementação de um modelo analítico e o desenvolvimento de processos ETL, à custa de uma menor capacidade na utilização de fontes de dados, que não

possuam capacidade de cálculo de indicadores, como por exemplo, ficheiros de texto ou folhas de cálculo.

9.1.2 Caso de estudo

Assuma-se, para efeito desta análise a existência de uma organização que necessita de melhorar as suas ferramentas de gestão de forma a optimizar o seu desempenho e, para esse efeito, inicia a construção de uma solução de *business intelligence* que permita mostrar num *dashboard web*, um conjunto de indicadores de desempenho sobre diferentes áreas de gestão. Nesta organização, é disponibilizado aos utilizadores um ambiente *Windows* e o *dashboard* com os indicadores terá de ser consultado pela equipa de gestão. Os dados do negócio estão armazenados em dois repositórios, um com dados financeiros e de contabilidade em *Oracle* e outro com dados operacionais em *Microsoft SQL Server* e dos quais deverão ser extraídos indicadores de desempenho.

Assuma-se ainda que, três meses após o início do projecto, a organização passa por um processo de fusão com outra entidade que tem os seus dados de negócio armazenados numa plataforma *Oracle E-Business Suite*. A solução deverá ter a capacidade de integrar os dados provenientes desta terceira fonte de dados e o *dashboard* terá de estar acessível a um conjunto de novos elementos da equipa de gestão provenientes da nova entidade.

9.1.3 Custos de implementação

Os custos de implementação da solução pretendida no cenário acima descrito serão analisados considerando e comparando a utilização do modelo tradicional através de tecnologias Oracle ou Microsoft com o modelo ágil através da ferramenta proposta, sendo tidos em linha de conta três factores: **custos de desenvolvimento**, **custos da infra-estrutura de suporte** e **custos com licenciamento de tecnologias**.

1. **Custos do desenvolvimento:**

Como foi visto na secção **9.1.1** , o processo de desenvolvimento de soluções *business intelligence* tradicionais, sendo mais generalista, é bastante complexo, no sentido em que é constituído por várias fases, implicando a utilização e integração de diferentes ferramentas e tecnologias. Este factor leva a que, por um lado, a sua concretização envolva equipas multidisciplinares, constituídas por pessoas do foro funcional com conhecimento do negócio ou domínio em que está a ser aplicada a solução, e pessoas com conhecimentos das várias ferramentas necessárias e, por outro, a que ferramenta seja construída no

âmbito de um projecto de grande dimensão exigindo, consequentemente, um tempo de construção substancialmente elevado.

O desenvolvimento de uma solução ***TrueKPI*** pode ser concretizado com uma equipa mínima (no limite apenas uma pessoa), que não precisa de conhecimentos técnicos da tecnologia em que assenta a solução, bastando tomar conhecimento do negócio/domínio em causa e possuir alguma proficiência na utilização da linguagem *SQL* para escrever as *queries* que definem os indicadores de desempenho.

2. **Custos da infra-estrutura de suporte:**

O desenvolvimento tradicional de soluções de *business intelligence* carece muitas vezes de uma grande infra-estrutura para suportar os vários passos do processo, desde a extracção dos dados até a disponibilização da informação. Apesar de ser possível a utilização de configurações mínimas para suportar todo o fluxo de dados, uma solução com bom desempenho e escalabilidade passa por ter múltiplos servidores dedicados a cada um dos passos.

O desenvolvimento da solução através do ***TrueKPI***, necessita de apenas um servidor responsável por todos os passos do fluxo de dados

3. **Custos da infra-estrutura de suporte:**

As soluções tradicionais são construídas com recurso a diferentes ferramentas e tecnologias que são licenciadas em separado. Na Tabela 7 é apresentada uma lista típica de componentes para projectos análogos ao descrito, utilizando tecnologias Microsoft ou Oracle, na qual se classificam os componentes enumerados quanto ao seu custo de licenciamento

	Microsoft			**Oracle**	
	Tecnologia	**Licenciamento**		**Tecnologia**	**Licenciamento**
Extracção de dados	*Integration Services*	Integrante das versões SQL Server avançadas		*Transparent Gateway*	Moderado
Modelo Analitico	*Analysis Services*	Elevado		*Oracle RDBMS*	Muito Elevado
Front-End	*Performance Point*	Muito Elevado		*Discoverer*	Elevado
Máquinas	*Windows Server 2003*	Moderado		*Windows Server 2003 / Linux*	Moderado / Baixo

Tabela 7 – Custos de licenciamento para soluções clássicas

A abordagem ***TrueKPI*** permite disponibilizar a solução sem licenciamentos adicionais para além do respeitante ao sistema operativo do servidor que hospeda a aplicação.

9.1.4 Resiliência

Após a descrição dos processos e custos de desenvolvimento de uma aplicação de *business intelligence*, seguindo o modelo clássico e seguindo o modelo proposto no âmbito deste trabalho, esta secção pretende analisar a resposta que cada um modelos dá à segunda parte do cenário descrito – uma alteração do contexto de negócio em que a aplicação está a ser desenvolvida – e quais as implicações desta alteração.

No caso de estudo apresentado, a alteração no contexto de negócio traz dois problemas à construção da aplicação de *business intelligence*. Por um lado, passa a existir mais uma fonte de dados e, por outro, podem ocorrer alterações nos conceitos do negócio: podem surgir novos conceitos e outros podem ser alterados ou eliminados. Convém salientar que, provavelmente, passados três meses após o início do projecto, a aplicação construída através da abordagem clássica ainda estaria em desenvolvimento e durante o período em que os contornos da fusão não estivessem claros, o desenvolvimento seria suspenso. Ao invés, a aplicação construída através da ferramenta proposta já estaria disponível e poderia continuar a ser utilizada, ainda que parcialmente, durante o processo de fusão.

No caso da aplicação desenvolvida segundo a abordagem tradicional, estas alterações implicariam: redefinir ou adicionar processos *ETL* para extrair dados do novo repositório e alterar o modelo de dados analítico para considerar as alterações nos conceitos de negócio ou, em casos extremos, refazer todo o modelo. É igualmente de referir que o acesso de novos utilizadores ao sistema pode obrigar a revisões a nível de licenciamento, como por exemplo no caso de uma solução disponibilizada com tecnologia Oracle, em que o acesso à aplicação de interface ao sistema, é licenciada com base no número de utilizadores que a utilizam.

A resposta dada pela aplicação ***TrueKPI*** a estas alterações de contexto consistiria em acrescentar novas *queries* e refazer algumas das já existentes.

As respostas apresentadas para cada uma das arquitecturas podem igualmente ser extrapoladas para cenários em que, não ocorrendo alterações de contexto, é necessário acrescentar novos indicadores que não estavam previstos inicialmente.

9.1.5 Conclusões

O estudo apresentado neste capítulo, permite concluir que a implementação da solução descrita no cenário da secção 9.1.2 , seguindo a abordagem tradicional comporta custos bastante elevados a nível de recursos humanos, infra-estrutura, licenciamento e manutenção o que pode tornar esta opção proibitiva para muitas organizações. A complexidade inerente à construção destas aplicações faz com que, tipicamente, sejam concretizadas em projectos de grande dimensão e nesse sentido, sofram dos problemas associados a todos os projectos desta natureza: dificuldade em reagir a mudanças e o risco do resultado final estar desadequado ou obsoleto face ao momento em que o projecto foi iniciado. Ainda assim, as soluções desenvolvidas segundo esta abordagem apresentam melhor desempenho, maior escalabilidade, consomem menos recursos nas fontes de dados de negócio e, através da implementação de processos de *ETL*, suportam mais tipos de fontes de dados, sendo a melhor opção para casos em que:

- Existam grandes quantidades de dados;
- Os indicadores tenham de ser recalculados muito frequentemente e as *queries* que os concretizam sejam muito complexas;
- O modelo de dados origem seja pouco eficiente, devida a uma má organização, a uma complexidade muito elevada, ou por falta de estratégias de optimização de bases de dados.

As soluções criadas com a ferramenta ***TrueKPI*** seguem um paradigma mais ágil, o que faz com que seja mais resiliente a mudanças de requisitos ou de contexto, permitindo a produção de valor num período de tempo significativamente mais curto, melhorando o *breakeven* do investimento feito pelos seus potenciais clientes. Na realidade é possível mostrar, logo numa primeira reunião de apresentação da ferramenta alguns indicadores da organização do cliente, caso o modelo de dados seja conhecido como quando, por exemplo, assenta num *ERP* como a *Oracle E-Business Suite*. Este factor tem um impacto muito significativo do ponto de vista comercial e de confiança por parte do cliente. O facto de ter um processo de desenvolvimento simples e necessitar de uma infra-estrutura ligeira, faz com que tenha custos muito reduzidos tanto a nível de construção, como de manutenção das aplicações. Do ponto de vista tecnológico, o facto de não serem usados processos *ETL* nem um modelo analítico para extracção e armazenamento de informação faz com que a aplicação tenha uma infra-estrutura mais leve, mas também implica que esta esteja directamente exposta aos repositórios de dados de origem e sofra as

consequências de acessos ou modelos de dados ineficientes. Este factor pode muitas vezes ser atenuado com a existência de um modelo de dados intermédio sobre o qual são calculados os indicadores.

Apesar da arquitectura proposta apresentar uma escalabilidade e desempenho potencialmente menores, a abordagem ágil a sistemas de *business intelligence*, já provou ser frutuosa em alguns contextos, como no caso da aplicação sobre a qual incidiu o processo de reengenharia que este relatório documenta.

9.2 Comparação com SicGest

O sistema desenvolvido permite a construção de *dashboards web* de visualização de indicadores de negócio que oferecem todas as funcionalidades disponibilizadas pela aplicação sobre a qual incidiu o processo de reengenharia – o *SicGest* –, nomeadamente:

- Celeridade na obtenção dos resultados dos indicadores (resultados em *cache*)
- Diferentes representações para os resultados dos indicadores
    - Tabelas
    - Gráficos
- Navegação para outras aplicações que detalhem ou complementem determinado valor de indicador (enriquecendo os resultados com o filtro de pós-processamento apresentado em 7.4.3)

Além destas funcionalidades os *dashboards* ***TrueKPI*** oferecem uma melhor experiência de utilização graças à utilização de técnicas de *AJAX* e permitem ainda:

- Obter indicadores a partir de diferentes fontes de dados:
    - Diferentes bases de dados;
    - Diferentes tecnologias de bases de dados;
    - *Web services*;
- Enriquecer os resultados dos indicadores com filtros de pós-processamento configuráveis;
- Definir indicadores parametrizáveis;
- Relacionar e navegar entre indicadores;
- Personalizar o *dashboard:*
    - Quais os indicadores a visualizar;
    - Qual a representações gráficas para os diversos indicadores;
    - Qual sua disposição.

Capítulo 10

Conclusões e Trabalho Futuro

O Projecto em Engenharia Informática descrito neste documento teve por objecto, por um lado, a investigação e concretização de um processo de desenvolvimento de *software* que promove a qualidade e optimiza o custos produção reduzindo o esforço de detecção e depuração de erros. Por outro lado, consistiu na criação de uma plataforma que permite implementar, com custos reduzidos, soluções de extracção e exploração de indicadores de negócio em diferentes contextos, abordando e abraçando perspectivas e desafios de natureza diversa:

- Desafios tecnológicos, na análise e escolha das tecnologias adequadas para suporte ao processo de desenvolvimento e para a concretização do sistema proposto, obrigando à ponderação das suas mais-valias e limitações;
- Desafios técnicos, na utilização de metodologias, tácticas e padrões de desenho adequados. No emprego de técnicas avançadas de programação, até aqui desconhecidas para o aluno, como o carregamento e invocação dinâmica de componentes e a geração automática de código e, ainda na utilização de diferentes paradigmas de desenvolvimento (orientado a objectos e orientado a aspectos);
- Desafios de domínio e de negócio, no estudo de ferramentas de *business intelligence* existentes no mercado e na compreensão do modelo de dados de suporte às aplicações de negócio de um cliente da Truewind, que serviu de base para a extracção de indicadores de teste;
- Desafios comerciais, na criação de um sistema com viabilidade comercial e que permite consolidar e expandir a actividade da organização de acolhimento.

Atentando aos resultados obtidos, bem como à avaliação da ferramenta desenvolvida, descritos respectivamente nos Capítulos 8 - Resultados e 9 –

Avaliação e Estudo Comparativo, podemos concluir que os objectivos propostos a nível de processo e de produto foram atingidos com sucesso. Evidência disso, é o propósito de integrar a funcionalidade de cálculo de indicadores do ***TrueKPI*** com outro Projecto em Engenharia Informática em curso na Truewind que consistiu, em parte, no estudo da adequabilidade de modelos de desenvolvimento ágeis, recolhendo e analisando métricas de projectos em que esta metodologia foi aplicada.

Em acréscimo a estes objectivos, o trabalho desenvolvido permitiu ainda obter alguns resultados que, ainda que intangíveis, se revestem de grande valor, quer para o aluno, quer para a organização de acolhimento, contribuindo para a melhoria da sua base de conhecimento a nível de:

- Metodologia para documentar atributos de qualidade e arquitecturas de *software*;
- Tácticas e padrões de desenho que permitem alcançar determinados atributos de qualidade;
- Arquitectura de software criada (como artefacto reutilizável);
- Exemplo de utilização de *AOP* para criar funcionalidades transversais a diferentes módulos de um sistema.

A plataforma desenvolvida foi intencionalmente desenhada para permitir a extensão e evolução, sendo possível identificar alguns caminhos de desenvolvimento futuro, nomeadamente:

- Através da biblioteca de geração de gráficos usada (*Dundas Charts)* permitir ao utilizador, sobre os indicadores calculados, realizar diferentes agrupamentos e a visualização de diferentes séries de dados no mesmo gráfico (do mesmo ou de diversos indicadores);
- Possibilitar a definição de regras de formatação que permitam destacar valores importantes ou que cumpram determinados requisitos (por ex.: assinalar a vermelho valores fora de limites (configuráveis) para o indicador);
- Estender o mecanismo de registo de invocações parametrizadas para, segundo um conjunto de regras e heurísticas, melhorar o desempenho do sistema criando automaticamente novos indicadores (já com os parâmetros necessários);
- Utilizar a técnica de carregamento de *providers* de acesso a dados para extrair indicadores de fontes sem capacidades de computação como folhas de cálculo ou ficheiros de texto;

- Explorar melhor as potencialidades da *framework Spring .NET* quanto às funcionalidades de *dependency injection* e implementação de aspectos.

Bibliografia e referências

1. Extreme Programming: A Gentle Introduction. *Extreme Programming: A Gentle Introduction.* [Online] http://www.extremeprogramming.org/.

2. **McConnel, Steve.** *Code Complete.* s.l. : Microsoft Press, 2004.

3. Understanding Quality Attributes. *Software Architecture in Practice, Second Edition.* s.l. : Addison Wesley, 2003.

4. Patterns&Practices - proven practices for predictable results. *msdn.microsoft.com.* [Online] Microsoft. http://msdn.microsoft.com/en-us/library/ms978599.aspx.

5. **Christopher Alexander, Sara Ishikawa, Murray Silverstein.** *A Pattern Language: Towns, Buildings, Construction.* 1997.

6. ADO .NET. *msdn.* [Online] Microsoft. http://msdn.microsoft.com/en-us/library/e80y5yhx(vs.80).aspx.

7. ASP .NET Developer Center. *msdn.* [Online] Microsoft. http://msdn.microsoft.com/en-us/asp.net/aa336567.aspx.

8. Portlet. *Wikipedia.* [Online] Wikimedia Foundation. http://en.wikipedia.org/wiki/Portlet.

9. Web Parts. *msdn.* [Online] Microsoft. http://msdn.microsoft.com/en-us/library/e0s9t4ck.aspx.

10. **Fowler, Martin.** Inversion of Control Containers and the Dependency Injection pattern. *Martin Fowler.* [Online] http://martinfowler.com/articles/injection.html.

11. Lotus Notes - Business email solution. *IBM.* [Online] IBM. http://www-01.ibm.com/software/lotus/products/notes/.

12. Windows Sharepoint Services. *Microsoft Office Online.* [Online] Microsoft. http://office.microsoft.com/pt-pt/sharepointtechnology/FX100503842070.aspx.

13. What Software Architecture Is and What It Isn't. *Software Architecture in Practice, Second Edition.* s.l. : Addison Wesley, 2003.

14. The Agile Blog. [Online] Danube. http://www.danube.com/blog/michaeljames/user_story_examples_and_counter examples.

15. *Structured Design.* **W. Stevens, G. Myers, L. Constantine.** s.l. : IBM Systems Journal, 1974.

16. Software Product Lines: Re-using Architectural Assets. *Software Architecture in Practice, Second Edition.* 2003.

17. **Burnstein, Illene.** *Practical Software Testing.* New York : Springer Science+Business Media, 2003.

18. *On the Criteria to Be Used in Decomposing Systems Into Modules.* **Parnas, D.L.** s.l. : communications of the ACM, 1972.

19. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l. : Addison Wesley, 1995.

20. *Aspect Oriented Programming.* **Gregor Kiczales, John Irwin , John Lamping , Jean-Marc Loingtier , Cristina Videira Lopes , Chris Maeda , Anurag Mendhekar.** 1997.

21. All About Agile. [Online] http://www.agile-software-development.com/2008/01/user-stories-answers-on-postcard-please.html.

22. *A complexity measure.* **McCabe, Thomas J.** 4, 1976, Vols. SE-2.

23. Cruise Control .NET. *Cruise Control .NET.* [Online] thoughtworks. http://confluence.public.thoughtworks.org/display/CCNET/Welcome+to+Cruise Control.NET.

24. Scrum Alliance. *Scrum Alliance.* [Online] http://www.scrumalliance.org/.

25. NAnt Homepage. *NAnt - A .NET build tool.* [Online] http://nant.sourceforge.net/.

26. Nunit. *Nunit.* [Online] http://www.nunit.org.

27. NMock. *NMock: A Dynamic Mock Object Library.* [Online] http://www.nmock.org/.

28. NCover. *NCover - Code Coverage for .NET Developers.* [Online] http://www.ncover.com/.

29. Watin. *Watin .* [Online] http://watin.sourceforge.net/.

30. FxCop. *FxCop.* [Online] http://msdn.microsoft.com/en-us/library/bb429476%28VS.80%29.aspx.

31. Microsoft SQL Server 2005. *Microsoft.* [Online] Microsoft. http://www.microsoft.com/Sqlserver/2005/en/us/reporting-services.aspx.

Capítulo 11 Anexos

11.1 Glossário

1. AJAX – Assynchronous Javascript And XML
2. Assembly – Pedaço de código compilado na framework .NET
3. ADO .NET – Active Data Objects. Tecnologia Microsoft de acesso a dados. [6]
4. API – Application Program Interface.
5. ASP .NET – Active Server Pages. Tecnologia Microsoft de criação de páginas dinâmicas e aplicações Web. [7]
6. CRM – Customer Relationship Management
7. Connection String – Frase com todos os argumentos que especificam os parâmetros necessários para estabelecer uma ligação a um determinado repositório de dados.
8. COTS – Components Of The Shelf
9. Dashboard – painel de controlo ou visualização de informação.
10. ERP – Enterprise Resource Planning.
11. ETL – Extraction, Tranformation and Loading
12. FIFO – First In First Out
13. IDE – Integrated Development Environment
14. KPI – Key Performance Indicator.
15. Thread Pool – Conjunto de threads reutilizáveis, criadas de antemão de forma a aumentar o desempenho de aplicações paralelas, evitando o custo de criação e destruição de novas threads.
16. Portlet – ver [8]
17. SCM – Supply Chain Management
18. SLA – Service Level Agreement
19. SQL – Structured Query Language
20. SOA – Service Oriented Architecture
21. Thread – Fio de execução independente de um programa.

22. WebPart – Controlo de servidor na tecnologia ASP .NET que permite a renderização de HTML dinâmica dentro de uma página faça de hospedeiro. [9]
23. WebParts Page – página criada na tecnologia ASP .NET que funciona como hospedeira de webParts [9].
24. WSDL – Webservices Description Language

11.2 Padrões de desenho

11.2.1 *Abstract Factory*

O padrão de desenho *abstract factory*, *fornece uma interface para criar famílias de objectos relacionados ou dependentes sem especificar as suas classes concretas*[11], isolando-as e facilitando a permutação entre famílias de objectos e promovendo a consistência entre estas.

Foi utilizado, criando um elemento responsável por instanciar a classe de acesso ao repositório local, parametrizada com um dos componentes que implementa um algoritmo de gestão da *cache* (ver 7.5.1). Desta forma é possível mudar o componente que interage directamente com o repositório (no caso de ser necessário mudar a tecnologia deste) e/ou mudar o algoritmo de gestão de *cache*, isolando o impacto destas alterações no restante sistema.

Classes concretas envolvidas:

- *DataStoreFactory*
- *IValuesManager*
- *XMLValuesManager*
- *DynamicDataValuesManager*
- *IDataStore*
- *SQLiteDataStore*

11.2.2 *Dependency Injection*

Dependency Injection[11] consiste em fornecer as dependências externas de um componente e é uma especialização de um outro padrão de desenho conhecido como *Inversion of Control* (ver [10]). Neste caso, é feita a inversão do processo de obtenção das dependências, o que torna os componentes mais simples e promove o princípio de *low-coupling*.

Este padrão de desenho foi utilizado em vários pontos do sistema, nomeadamente aquando criação dos elementos *Worker* (o construtor deste elemento recebe uma lista de filtros de pós-processamento já instanciada, fazendo com que este não tenha de "saber" construir essa lista), ou na criação dos elementos *IDataStore* (que recebem um elemento *IValuesManager*, responsável pelo encapsulamento do algoritmo de gestão da *cache*, já instanciado e pronto a utilizar).

11.2.3 *Facade*

Este padrão de desenho *fornece uma interface unificada de alto nível para um conjunto de interfaces de um subsistema, tornando-o mais fácil de usar*[11]. Permite isolar os clientes dos vários componentes de um subsistema, reduzindo a sua complexidade, as dependências a nível de compilação (melhorando o processo de construção) e promovendo o *low-coupling* entre os clientes e o subsistema, a interoperabilidade deste último e a portabilidade do sistema como um todo.

Foi aplicado, criando um ponto de único de acesso e interacção com o sistema, materializado no componente *Manager*.

Classes concretas envolvidas:

- *TrueKPI Webservice*
- *Manager*

11.2.4 *Factory Method*

O padrão de desenho *factory method*, *define uma interface para a criação de um objecto, mas deixa que as classes invocadoras decidam qual a classe concreta que querem instanciar, diferindo assim o momento de instanciação*[11].

Este padrão foi utilizado para implementar o carregamento dinâmico de *providers* de acesso a dados, classes *proxy* para interagir com *webservices* e filtros de pós-processamento, disponibilizando elementos responsáveis pela criação destes objectos e abstraindo a forma como estes são construídos.

Classes concretas envolvidas:

- *DataAccessFactory*
- *IDataAccess*
- *DataBaseAccess*
- *WebserviceAccess*
- *PostProcessorFactory*
- *IPostProcessor*

- *PostProcessor*

11.2.5 *Proxy*

Este padrão de desenho fornece um objecto mediador que controla o acesso a outro componente, criando assim um nível de indirecção entre o elemento invocador e o invocado. Esta abordagem permite "esconder" determinadas particularidades do objecto invocado, como por exemplo, a sua localização, implementação e complexidade, oferecendo uma interface simples aos elementos invocadores.

Foi utilizado na camada cliente, para concretizar um ponto de acesso aos *webserices*, "escondendo" as peculiaridades da invocação remota.

11.2.6 *Singleton*

Este padrão de desenho permite assegurar que uma classe só tem uma instância, no sentido de obter maior controlo no acesso a esta, ou implementar uma semântica específica.

Foi usado para garantir que só existe uma instância da classe *Manager*, assegurando assim a consistência das operações e controlo no acesso à camada lógica.

Classes concretas envolvidas:

- *Manager*

11.2.7 *Strategy*

O padrão de desenho *strategy* permite *definir uma família de algoritmos e encapsular cada um deles de forma a que sejam permutáveis. Este padrão permite que os algoritmos possam evoluir independentemente dos clientes que o utilizam*[11].

Foi contemplado para encapsular os dois algoritmos desenhados para gestão da *cache* de valores pré-calculados para os indicadores.

Classes concretas envolvidas:

- *IValuesManager*
- *DynamicDataValuesManager*
- *XMLValuesManager*

11.3 Medições dos algoritmos de gestão de cache

	Ensaio 1	Ensaio 2	Ensaio 3	Ensaio 4	Ensaio 5	Ensaio 6	Ensaio 7	Ensaio 8	Ensaio 9	Ensaio 10	Average
XML 1	0,851224	0,6609504	0,5908496	0,6609504	0,9313392	1,73249212	0,6809792	0,5808352	0,6309072	0,6208928	0,794142012
XML 10	1,0414976	0,75108	0,7210368	1,051512	0,9213248	0,9213248	0,951368	0,7911376	0,68909792	0,7610944	0,860047392
XML 100	4,0658464	4,7768688	4,1860192	4,4964656	4,0658464	5,557992	4,6667104	4,556552	5,107344	3,955688	4,54353328
XML 1000	39,4767648	42,3008256	39,4667504	38,405224	39,5869232	38,9660304	38,0146624	40,2979456	38,3150944	38,1949216	39,30251424
XML Thread 1	0,6309072	0,5207488	0,4907056	0,4706768	0,7410656	0,5908496	0,3905616	0,550792	0,4105904	0,4105904	0,5207488
XML Thread 10	0,480691	0,5207488	0,5708208	0,5107344	0,6709648	0,4907056	0,3605184	0,4606624	0,3605184	0,4306192	0,48569838
XML Thread 100	0,3805472	0,600864	0,4806912	0,7811232	0,5808352	0,400576	0,4105904	0,4907056	0,4406336	0,5407776	0,5107344
XML Thread 1000	0,55608064	0,8311952	0,5307632	0,5207488	0,6609504	0,8111664	0,8912816	0,550792	0,7210368	0,6309072	0,670492224
Dinamic Data 1	0,4706768	0,550792	0,5407776	0,5708208	0,4206048	0,50072	0,5708208	0,5307632	0,4706768	0,3705328	0,49971856
Dinamic Data 10	0,9613824	0,7610944	0,6309072	0,6709648	0,851224	0,8912816	1,1416416	0,8612384	1,3619584	1,1115984	0,92432912
Dinamic Data 100	11,4464592	12,3477552	15,6825504	16,2433568	11,4064016	12,167496	11,917136	12,2976832	10,4049616	19,3077632	13,32215632
Dinamic Data 1000	126,0512528	157,4964688	122,7465008	117,3387248	133,0713472	125,8008928	124,8995968	122,5662416	115,66632	125,981152	127,1618498
Dinamic Data Thread 1	0,7911376	0,5207488	0,400576	0,4406336	0,4306192	0,3805472	0,4806912	0,3805472	0,3905616	0,3404896	0,4556552
Dinamic Data Thread 10	0,5407776	0,4306192	0,4306192	0,400576	0,550792	0,4806912	0,5107344	0,4706768	0,4105904	0,4406336	0,46667104
Dinamic Data Thread 100	0,450648	0,450648	0,4806912	0,50072	0,4706768	0,5608064	0,5207488	0,4606624	0,4806912	0,600864	0,49771568
Dinamic Data Thread 1000	0,550792	0,5708208	0,5307632	0,8412096	0,4706768	0,450648	0,4706768	0,4606624	0,550792	0,6909936	0,55880352

	Ensaio 1	Ensaio 2	Ensaio 3	Ensaio 4	Ensaio 5	Ensaio 6	Ensaio 7	Ensaio 8	Ensaio 9	Ensaio 10	Average
XML	3,6352272	3,655256	4,1259328	3,765144	4,055832	3,9256448	3,7153424	3,625128	3,6352272	3,855544	3,79942784
Dinamic Data	1,1816992	1,1416416	1,0715408	1,1416416	1,2217568	1,2217568	1,0915696	1,22117424	1,151656	1,1115984	1,155603504

Tabela 8 – Ensaios de desempenho dos algoritmos de gestão de cache (segundos)

11.4 Sumário de componentes

			Fases do fluxo de informação			
Camada	**Componente**	**Descrição**	**Criar Gerir**	**Obter**	**Armazenar**	**Visualizar**
Lógica	*AddURL*	Filtro de pós processamento que acrescenta aos resultados do indicador um url para uma aplicação externa		**X**		
	DataAccess Factory	Constrói componentes do tipo IDataAccess				
	Database Access	Classe abstracta para extracção de indicadores em bases de dados		**X**		
	DataStore Factory	Constrói elementos do tipo IDataStore				
	DynamicData Values Manager	Encapsula um algoritmo de gestão da cache baseado num modelo de dados dinâmico			**X**	**X**
	Engine	Responsável por orquestrar o processo de obtenção de indicadores		**X**	**X**	**X**
	Format Currency	Filtro de pós-processamento que formata o valor de coluna para um formato monetário		**X**		
	IDataAccess	Interface para os elementos que tem a capacidade extrair dados de diferentes fontes		**X**		
	IDataStore	Interface os para componentes responsáveis por interagir com o repositório local	**X**		**X**	**X**
	IPost Processor	Interface que define um filtro de pós-processamento de resultados dos indicadores		**X**		
	Ivalues Manager	Interface para componentes que encapsulam algoritmos de gestão da cache			**X**	**X**
	Manager	Faz a gestão do sistema e concretiza a interface com o exterior	**X**			**X**

	Componente	Descrição	Criar Gerir	Obter	Armazenar	Visualizar
	Post Processor	Classe abstracta que deve ser estendida para implementar filtros de pós-processamento		**X**		
	Post Processor Factory	Constrói elementos do tipo IPostProcessor				
	SQLite DataStore	Implementação do interface IDataStore para comunicação com um repositório SQLite	**X**		**X**	**X**
	Webservice	Implementa o webservice de acesso á camada lógica	**X**			**X**
	Webservice Access	Classe abstracta para extracção de indicadores em webservices		**X**		
	Worker	Responsável por incorporar um determinado indicador e obter os valores para esse indicador		**X**		**X**
	XMLValues Manager	Encapsula um algoritmo de gestão da cache baseado em serialização/desserialização dos resultados no formato xml			**X**	**X**
	Sharepoint Webpart	Componente com as mesmas responsabilidades do TrueKPI Webpart mas adaptado para funcionar em ambiente Sharepoint				**X**
Cliente Web	*TrueKPI Webpart*	Representa um indicador e oferece métodos de visualização e interacção com esse indicador				**X**
	TrueKPIService	Proxy para o *webservice* da camada lógica				**X**
	Webservice Manager	Abstrai o proxy da camada lógica				**X**
	Cache Manager	Abstrai os serviços de gestão da cache local				**X**
Cliente Mobile	*DataManager*	Ponto único de acesso aos dados				**X**
	IChartProvider	Interface para classes que geram representações gráficas para os indicadores				**X**

Componente	Descrição	Criar Gerir	Obter	Armazenar	Visualizar
Main Form	Form inicial da aplicação				**X**
Resco Compact Chart	Implementa a interface IChartProvider e usa as funcionalidades de geração de gráficos da biblioteca Resco Compact Charts				**X**
ShowKPI Form	Form de visualização de indicadores				**X**
TrueKPI Webserice Proxy	Proxy para o *webservice* da camada lógica				**X**
Webservice Manager	Abstrai o proxy da camada lógica				**X**

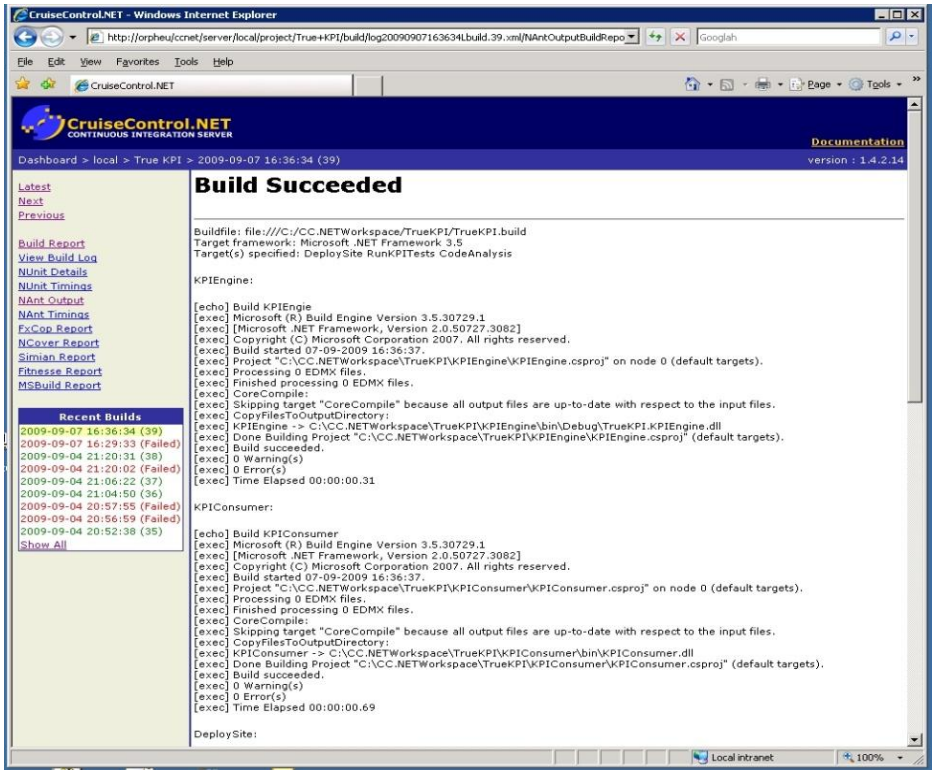
Tabela 9 – Sumário de componentes da arquitectura de software

11.5 Exemplo de configuração de aspectos

```
<spring>
    <objects xmlns="http://www.springframework.net">
      <object id="kpiLoggerAdvisor"
type="Spring.Aop.Support.NameMatchMethodPointcutAdvisor,
Spring.Aop">
        <property name="Advice">
          <object
type="TrueKPI.KPIEngine.Core.ParametrizedKPILoggerAdvice,
TrueKPI.KPIEngine" />
        </property>
        <property name="MappedNames">
          <list>
            <value>showFilteredKPI</value>
          </list>
        </property>
      </object>

      <object id="exceptionsAdvice"
   type="TrueKPI.KPIEngine.Core.ExceptionsHandlerAdvice,
TrueKPI.KPIEngine" />

      <object id="Manager"
type="Spring.Aop.Framework.ProxyFactoryObject">
        <property name="Target">
          <object type="TrueKPI.KPIEngine.Core.Manager,
TrueKPI.KPIEngine" />
        </property>
        <property name="InterceptorNames">
          <list>
            <value>kpiLoggerAdvisor</value>
            <value>exceptionsAdvice</value>
          </list>
        </property>
      </object>

    </objects>
  </spring>
```

11.6 Exemplos de utilização

11.6.1 Servidor de Integração



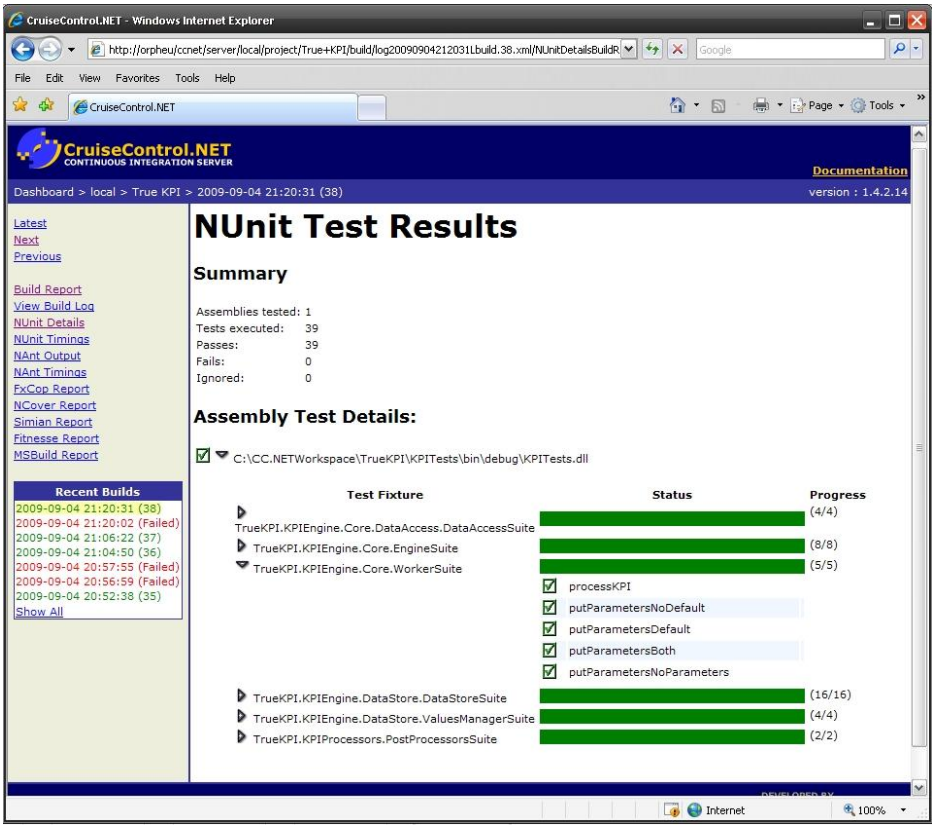
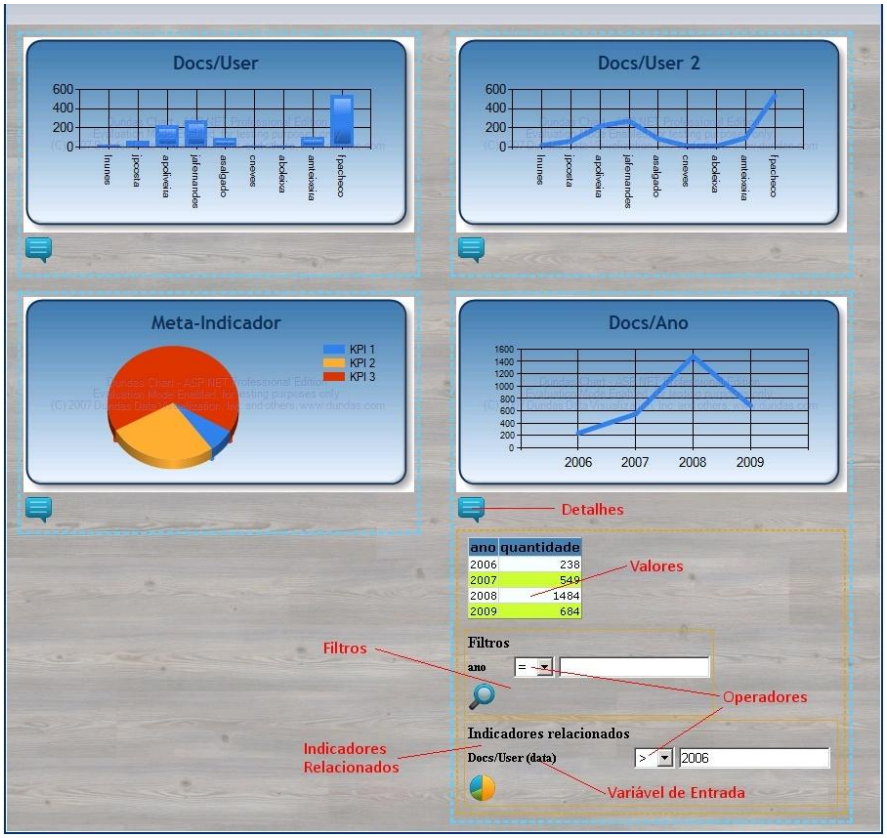


Figura 29 - Relatório *nAnt* e relatório *nUnit*

11.6.2 Família *TrueKPI*

TrueKPI Dashboard



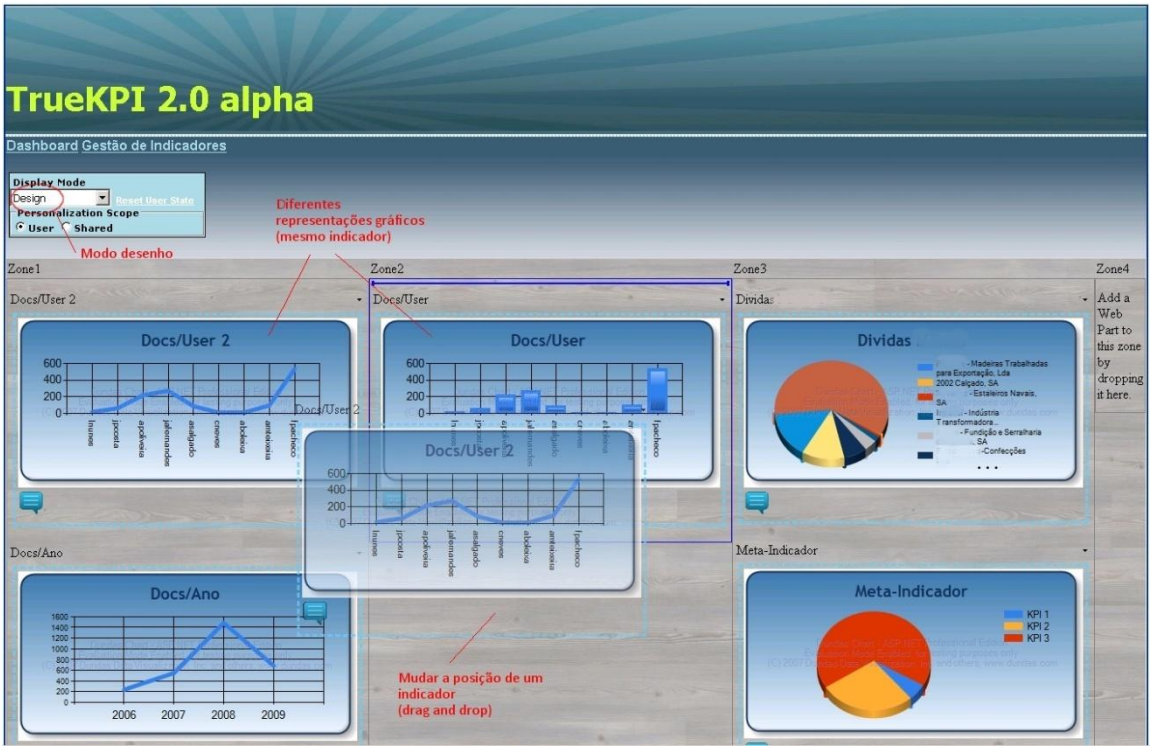
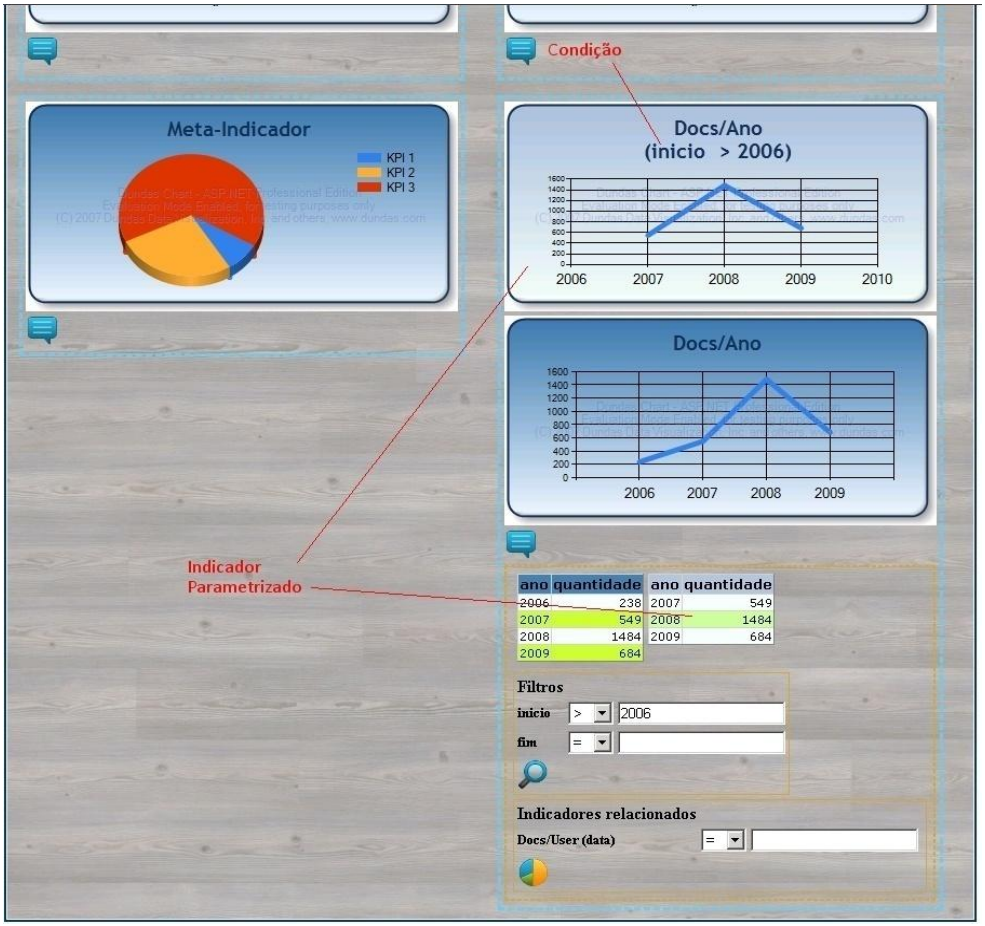


Figura 30 - *Dashboard* (detalhes e personalização)



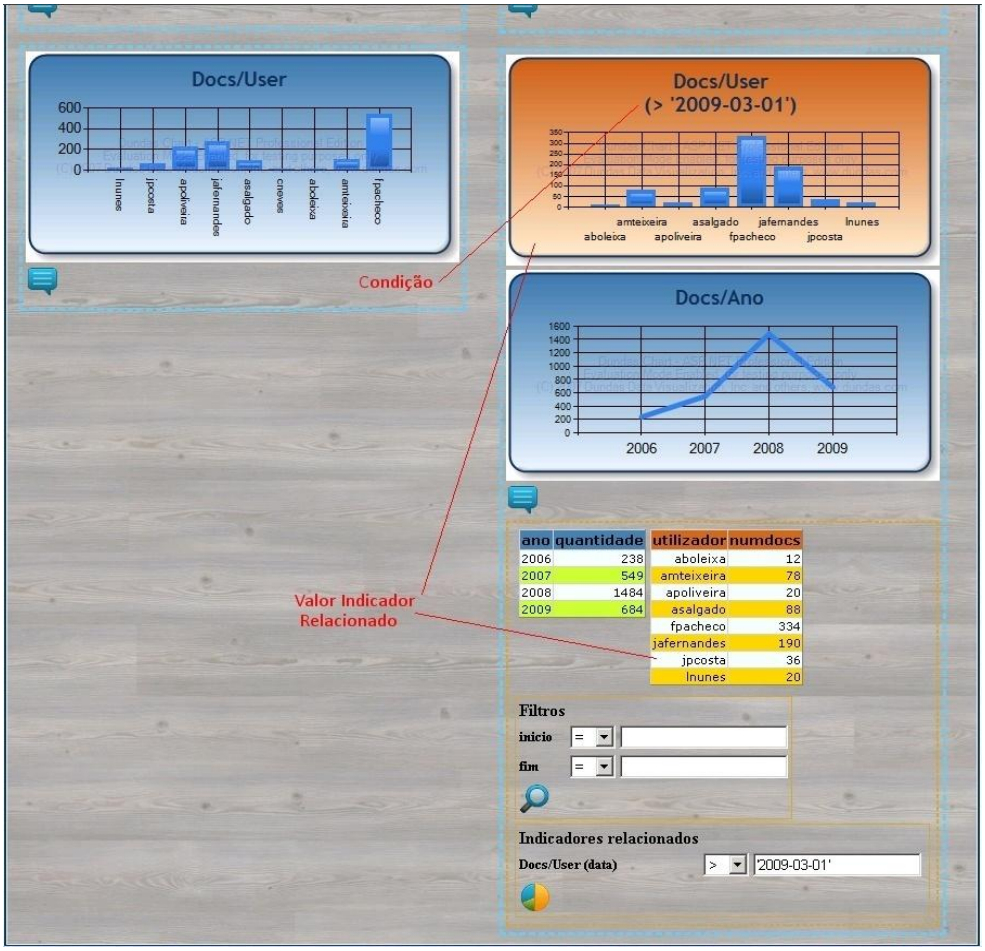
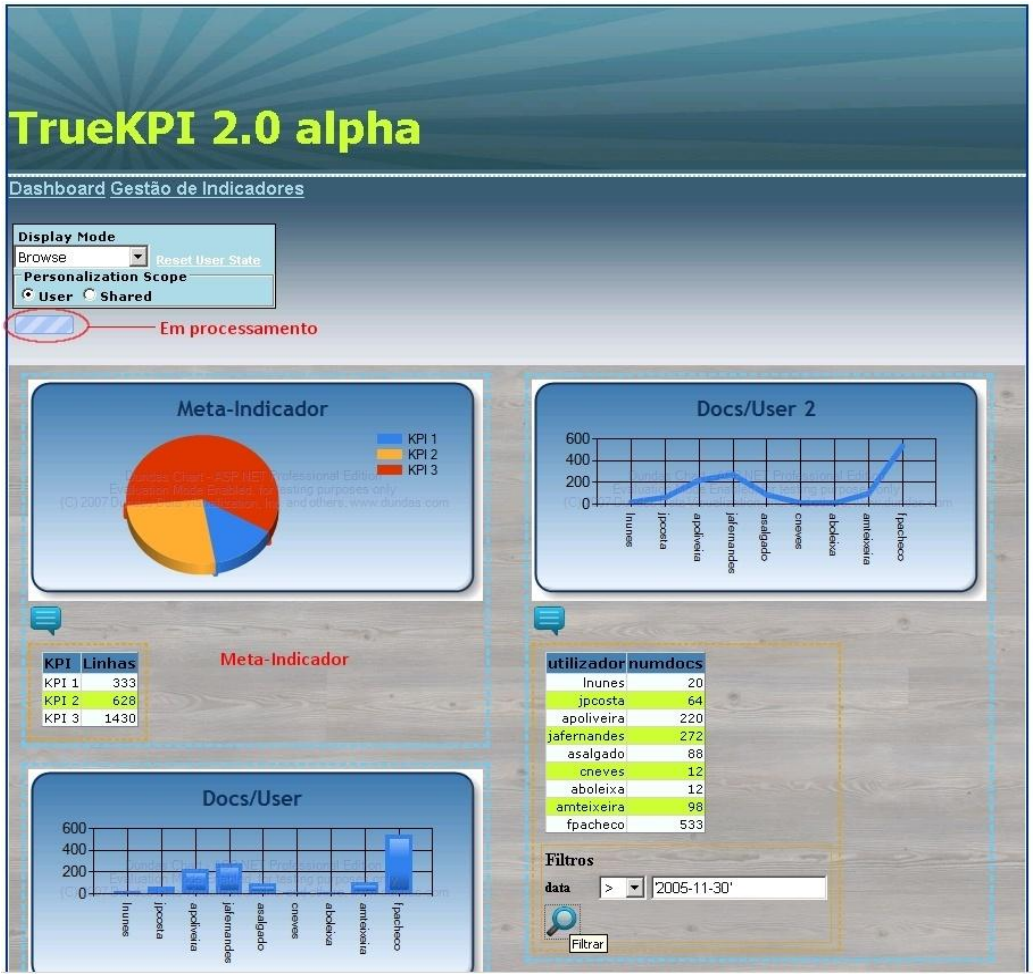


Figura 31 - *Dashboard* (Indicador parametrizado e indicadores relacionados)



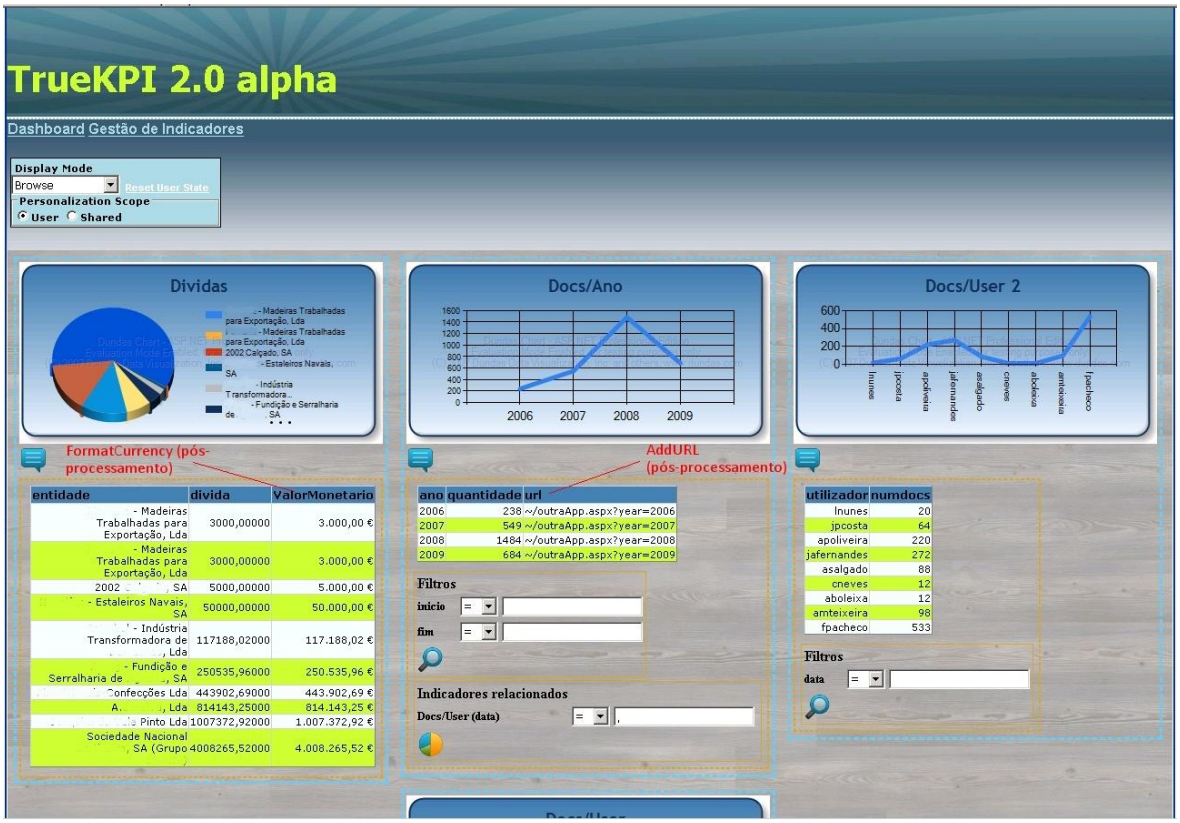


Figura 32 - *Dashboard* (em processamento e filtros pós-processamento de indicadores)

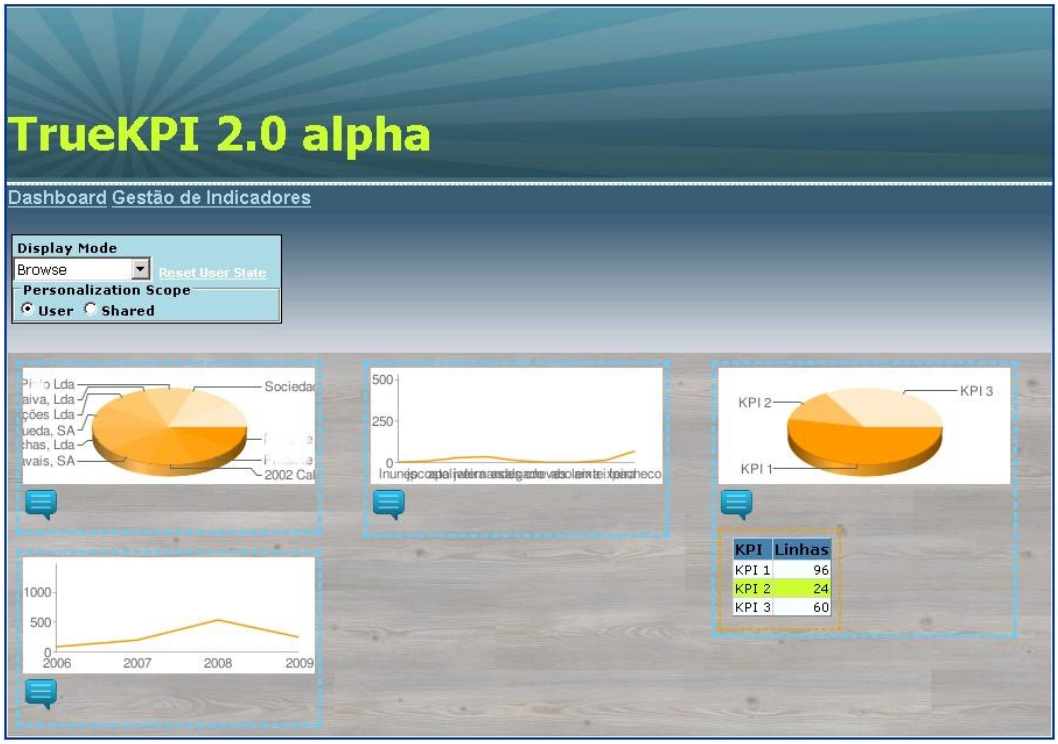


Figura 34 - *Dashboard* (Gráficos gerados através do *Google Charts*)

Figura 33 – Indicadores para erros e invocações parametrizadas

TrueKPI Mobile

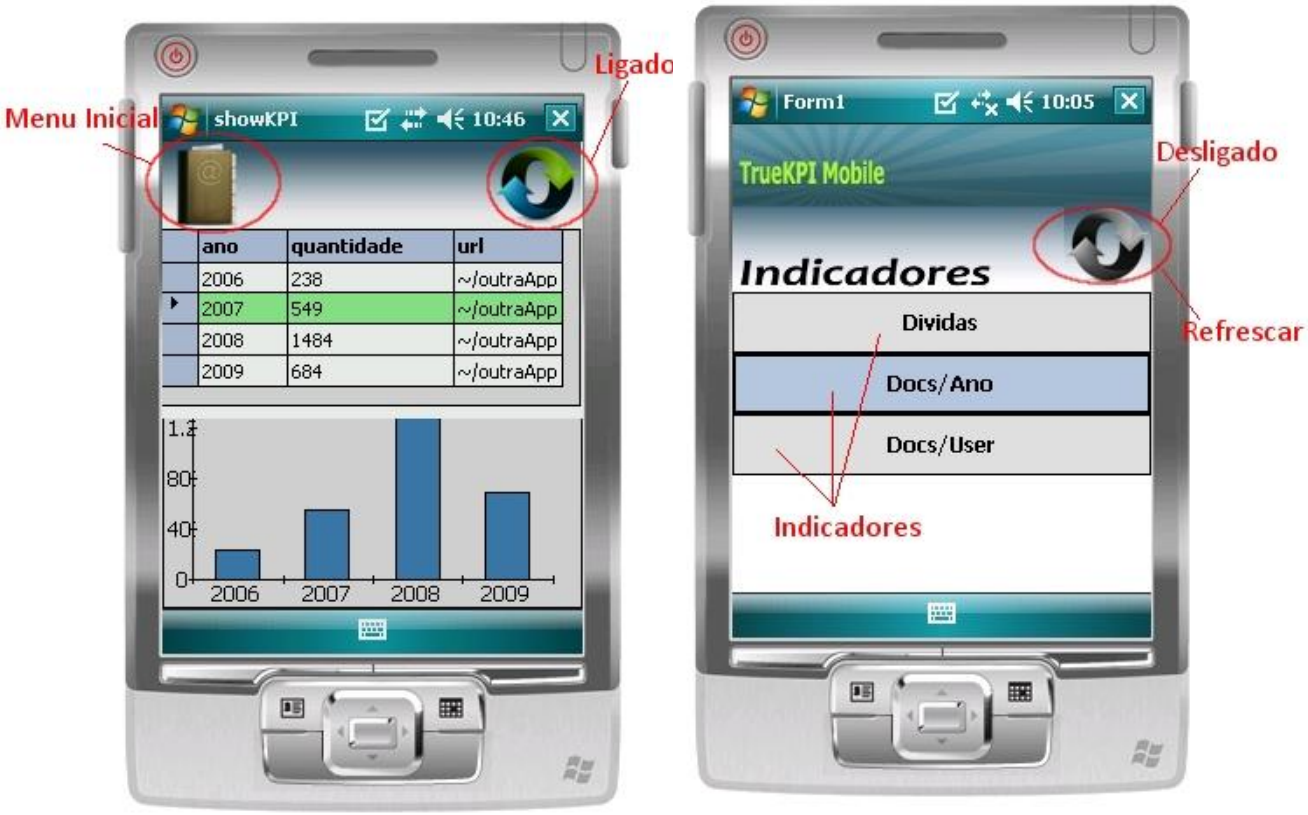


Figura 35 - ***TrueKPI Mobile***

11.7 Outros Projectos

O estágio realizado permitiu ao aluno, não só concretizar o projecto proposto, mas também participar em projectos reais para alguns clientes da empresa. Estes projectos permitiram tomar contacto com outras tecnologias, desenvolver capacidades de gestão de projectos, contactar e interagir com clientes e em especial usar na prática algumas das técnicas e ferramentas que estudou e aplicou no âmbito deste trabalho. Nesta secção descrevem-se os projectos em que o aluno participou, os contributos mais relevantes para esses projectos e as tecnologias, técnicas e ferramentas utilizadas.

11.7.1 Tecnologias

Microsoft Windows Workflow Foundation

A *Windows Workflow Foundation* é a tecnologia da *Microsoft* para a criação, execução e gestão de processos de negócio ou fluxos de trabalho (*workflows)* integrando a *.NET framework* a partir da versão 3.0.

Microsoft Sharepoint Technologies

O *Sharepoint* é uma tecnologia que oferece as funcionalidades de criação de portais corporativos, intranets colaborativas e aplicações *web* de gestão documental, gestão de conteúdos e automatização de processos de negócio. Disponível na plataforma *Microsoft Office Sharepoint Server* (*MOSS*), que disponibiliza um grande número de *templates*, controlos e funcionalidades avançadas de suporte ao negócio, bem como ferramentas de integração com outras tecnologias e ainda na sua versão base – *Windows Sharepoint Services* (*WSS*) - que oferece apenas as funcionalidades e controlos básicos para a criação de aplicações e portais *Sharepoint*. O *MOSS* tem diferentes versões com diferentes custos de licenciamento, sendo que o *WSS* é gratuito.

Microsoft SQL Server Reporting Services

A *SQL Server Reporting Services* da *Microsoft*, é uma ferramenta de extracção de informação de bases de dados e de geração de relatórios, suportando diferentes formatos de apresentação, como o *HTML*, *Excel* e *PDF* entre outros.

Microsoft .Net Windows Forms

A tecnologia de *Windows Forms* é parte integrante da plataforma de desenvolvimento da *Microsoft* – *.NET Framework*, permitindo a criação de aplicações ricas que executam em ambientes Windows sobre uma máquina virtual, denominada *CLR* (*Common Language Runtime*) que oferece uma camada de abstracção entre o sistema operativo e as aplicações finais.

Oracle E-Business Suite

A *Oracle E-Business Suite* é um conjunto de aplicações de *ERP*, *CRM*, *SCM* e outros módulos de suporte ao negócio que assentam na tecnologia de bases de dados relacionais *Oracle RDBMS*, utilizando a plataforma *Oracle Application Server*, sendo constituída por diferentes linhas de produtos e módulos que permitem gerir e integrar toda a informação de diferentes eixos do negócio e oferecer diferentes vistas e funcionalidades sobre esta informação.

11.7.2 Projectos

Workflows de suporte a processos burocráticos internos

Foram redesenhadas, para um cliente na área da indústria, três aplicações de *workflows* baseadas na plataforma *Lotus Notes* [11], utilizando as tecnologias *Windows Workflow Foundation* e *Windows Sharepoint Services*.

Envolveu o desenvolvido de um portal de gestão de processos englobando as três aplicações de *workflow* de suporte aos seguintes processos:

- ***Pricing*** - definição de tabelas de preços de produtos para um cliente;
- ***Fornecedores*** – pedidos de abertura e alteração de fornecedores de matérias, materiais e de serviços;
- ***Clientes*** – pedidos de abertura e alteração de dados de clientes;

A participação do aluno neste projecto permitiu-lhe assumir a responsabilidade pela análise de requisitos, construção e testes das aplicações. Foi ainda responsável pela instalação e manutenção das aplicações, tendo participado na gestão do projecto, interagindo directamente com os utilizadores finais. O aluno contou com o apoio de um elemento sénior experiente na tecnologia *Sharepoint*.

Seguimento de Acções Industriais

Aplicação *Sharepoint*, desenvolvida em *Windows Sharepoint Services* para registo de incidentes num ambiente industrial, permitindo a documentação das medidas tomadas e a avaliação das acções de resolução.

O aluno foi responsável pela análise de requisitos, implementação e testes das aplicações. Foi ainda responsável pela instalação e manutenção das aplicações, bem como pela gestão do projecto e mediação das comunicações e interacções com os clientes.

Portal Colaborativo

Criação de um portal colaborativo de suporte a uma iniciativa desenvolvida por um instituto público com uma rede de parceiros, para financiamento de ideias e projectos, utilizando a tecnologia *Microsoft Office Sharepoint Server*. O portal tem como desígnio receber a submissão de propostas de financiamento para ideias e projectos e servir de plataforma comum para gestão de informação, colaboração e comunicação entre todos os intervenientes

no processo, nomeadamente: empresas, gestores, promotores, consultores e o próprio instituto.

O aluno foi integrado no projecto com objectivo de desenvolver alguns componentes para a gestão de informação de empresas, gestores e promotores de ideias e para a submissão de novas ideias e projectos no portal. Participou ainda na refactorização de algumas funcionalidades, na integração de *layouts* de apresentação e ainda na criação de relatórios utilizando os *Reporting Services da Microsoft*.

Módulo de vendas *offline*

Para um cliente com uma instalação da *Oracle E-Business Suite* (módulo *Accounts Receivable*) que suporta o canal de vendas constituído por um conjunto de distribuidores locais e que é acedida por utilizadores remotos associados a esses distribuidores e que se encontram espalhados por todo o território nacional, foi desenvolvida uma ferramenta de suporte a este canal de vendas. Tendo como objectivo a melhoria deste acesso, criou-se uma infra-estrutura e um conjunto de aplicações que permitem aos utilizadores trabalhar localmente de forma desligada e periodicamente sincronizar os dados com a *Oracle E-Business Suite*.

O aluno contribuiu com o desenvolvimento de um protótipo para o ecrã de processamento de recebimentos em tecnologia *Microsoft .NET Windows Forms* e na construção de *queries SQL* sobre tabelas *da Oracle E-Business Suite*.

11.7.3 Conhecimentos adquiridos

Nos projectos em que participou, o aluno, teve oportunidade de aplicar, de forma prática, algumas das metodologias e ferramentas relevantes no contexto do Projecto de Engenharia Informática, que servindo para consolidar conhecimentos, contribuíram para o desenvolvimento do sistema pretendido, nomeadamente:

- *Scrum*
- Integração contínua
- Testes unitários
- *Framework .NET*
- *Sharepoint*
- *Oracle E-Business Suite*